Extremely Low and Variable Bandwidth Image Compression with Region of Interest Applied to Real Time Underwater Robotic Interventions

by

Eduardo Moscoso Rubino

supervised by

Raul Marín Prades, Pedro Sanz Valero, Alberto José Álvares

February 2018

Doctoral Programme in Computer Science

Universitat Jaume I Doctoral School

Extremely Low and Variable Bandwidth Image Compression with Region of Interest Applied to Real Time Underwater Robotic Interventions

Report submitted by Eduardo Moscoso Rubino in order to be eligible for a doctoral degree awarded by the Universitat Jaume I

Eduardo Moscoso Rubino

Raul Marín Prades

Castellon de la Plana, February 2018

Declaration of Authorship

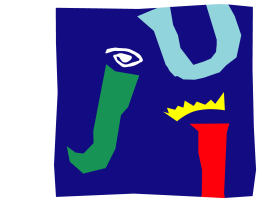
I, EDUARDO MOSCOSO RUBINO, declare that this thesis titled, "Extremely Low and Variable Bandwidth Image Compression with Region of Interest Applied to Real Time Underwater Robotic Interventions" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

February 2018

"If a man never contradicts himself, the reason must be that he virtually never says anything at all."

Erwin Schrödinger



Abstract

A new fast and progressive set-partitioning image compression parallel algorithm with Region Of Interest (ROI) which outputs an embedded bit oriented rate-distortion optimized stream and addresses very low bit rate compression is presented.

User defined variable packet sizes make it suitable for the implementation of any communications protocols, either underwater or in any other scenario, while remaining competitive with current state-of-the-art compressors at higher bit rates.

A parallel algorithm for the Discrete Wavelet Transform (DWT) based on the lifting scheme is also presented and it is shown to be optimal in the sense that no other implementation may be faster if memory saturation is achieved.

The best ordering for the significant and refinement bits of the transform coefficients is derived, using the Mean Squared Error (MSE) as the error measure, by fitting a Probability Density Function (PDF) to the transform coefficients and weighting the error for each range of coefficients by its respective DWT subband gain.

A general scheme for Region Of Interest (ROI), including a non-linear scaling ROI, is presented in which the lower bitplanes of the foreground coefficients are delayed in exchange for better background reconstruction, achieving a more effective blending of foreground and background information.

Finally, an implementation for both 32-bit and 64-bit ARM and x86 architectures was validated in an actual wireless underwater robotic teleoperation context.

Acknowledgements

My sincere thanks to all who engaged in conversations and contributions that led to the work presented here, in special Alberto Álvares (advisor), Raul Marín (advisor), Pedro Sanz (advisor), and Darius Burschka.

Special thanks also go to Humberto Abdalla who made me realize I had some inclination for scientific research and to Henrique Malvar who introduced me to the world of signal processing.

Finally, special thanks to Leonardo Rubino for existing and making life so much more interesting.

Contents

Declaration of Authorship iii

Abstract v

Acknowledgements vi

List of Figures xi

List of Tables xv

Abbreviations xvii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Context . . . 2
- 1.3 Available Solutions . . . 6
- 1.4 Proposed Solution . . . 9
- 1.5 Contributions . . . 12
- 1.6 Organization . . . 13

2 Related Work 15
- 2.1 Image Compression and Transmission and Underwater Robotics . . . 15
- 2.2 Image Compression . . . 22
- 2.3 Parallelization Schemes for the DWT . . . 25

3 Minimal Time DWT Algorithm 29
- 3.1 Introduction . . . 30
- 3.2 Cache Memory Architecture Overview . . . 32
- 3.3 Memory Cost Model . . . 34
- 3.4 DWT Lifting Algorithms . . . 35
  - 3.4.1 Vertical Naive Algorithm - cnaive . . . 37
  - 3.4.2 Horizontal Naive Algorithm - lnaive . . . 38
  - 3.4.3 Pipelined Algorithm - pipeline . . . 40
- 3.5 DWT Parallel Lifting Algorithms . . . 40
  - 3.5.1 Parallel Naive Algorithms . . . 41
  - 3.5.2 The Parallel Pipelined Algorithm - paraline . . . 41

3.6 Benchmark and Results . . . 47
3.6.1 64-bit x86 Dual Memory Channel . . . 48
3.6.1.1 STREAM Benchmark . . . 48
3.6.1.2 Bandwidth Saturation Index . . . 48
3.6.1.3 Performance Counters . . . 50
3.6.2 32-bit ARM SBC . . . 50
3.6.2.1 STREAM Benchmark . . . 51
3.6.2.2 Integer DWT . . . 53
3.6.2.3 Performance Counters . . . 55
3.6.2.4 Floating Point DWT . . . 55
3.7 Summary . . . 57

4 Data Prioritization **59**
4.1 Introduction . . . 60
4.2 Probability of an Interval . . . 62
4.3 Distortion . . . 62
4.4 Reconstruction Value . . . 63
4.5 Distortion Reduction per BIT . . . 63
4.5.1 Significant BIT . . . 63
4.5.2 Refinement BIT . . . 64
4.6 Entropy per BIT . . . 65
4.6.1 Significant BIT . . . 65
4.6.2 Refinement BIT . . . 66
4.7 Distortion Reduction per Bit . . . 68
4.7.1 Significant BIT . . . 68
4.7.2 Refinement BIT . . . 68
4.8 The Exponential Power Distribution . . . 69
4.8.1 Significant Entropy . . . 70
4.8.2 Significant Distortion Reduction . . . 71
4.8.3 Refinement Entropy . . . 72
4.8.4 Refinement Distortion Reduction . . . 73
4.8.5 Distortion Reduction Ratio . . . 74
4.9 Uniform Distribution . . . 77
4.10 Laplace Distribution . . . 78
4.11 Shape parameter estimation . . . 80
4.12 Examples . . . 82
4.13 Summary . . . 84

5 The Depth Embedded Block-Tree (DEBT) Algorithm **87**
5.1 Representation of variable depth blocks and trees . . . 90
5.2 Set Partition and Decomposition . . . 95
5.3 Lists of Sets and Refinement Bits . . . 97
5.4 List Traversal . . . 99
5.5 Algorithm Outline . . . 102
5.6 Set Partition and Decomposition Algorithm . . . 103
5.7 Examples . . . 106
5.8 Summary . . . 117

6 Region Of Interest **119**
6.1 Introduction . . . 120
6.2 ROI types . . . 121
6.2.1 Map ROI . . . 121
6.2.2 Bitplane ROI . . . 123
6.3 Examples . . . 124
6.4 Summary . . . 127

7 Case Studies and Results **129**
7.1 Paraline Algorithm . . . 129
7.1.1 32-bit ARM SBC . . . 130
7.1.2 64-bit x86 Dual Memory Channel Comparison . . . 132
7.2 Classic Images . . . 134
7.2.1 Airplane . . . 137
7.2.2 Baboon . . . 139
7.2.3 Barbara . . . 141
7.2.4 Lena . . . 143
7.2.5 Peppers . . . 145
7.2.6 Sailboat . . . 147
7.2.7 DEBT × JPEG-2000 Compression Comparison . . . 149
7.3 High Resolution Images . . . 150
7.3.1 Bridge . . . 151
7.3.2 Cathedral . . . 153
7.3.3 Moon . . . 155
7.3.4 Tree . . . 157
7.4 Underwater Images . . . 159
7.4.1 EPD Shape Parameter . . . 159
7.4.2 DEBT × JPEG-2000 Compression Comparison . . . 160
7.4.3 DEBT × JPEG-2000 Timing Comparison . . . 163
7.4.4 DEBT Lossless Timing and Compression Comparison . . . 164
7.5 Region Of Interest . . . 165
7.5.1 32-bit ARM Implementation Performance . . . 166
7.5.2 DEBT × JPEG-2000 Compression Comparison . . . 169

8 Conclusions **171**
8.1 Contributions . . . 171
8.2 Suggestions and Future Work . . . 173

A DWT Subband Gain Tables **185**
A.1 The 5/3 Integer Discrete Wavelet Transform . . . 185
A.2 The 9/3 Integer Discrete Wavelet Transform . . . 186
A.3 The 9/7 Integer Discrete Wavelet Transform . . . 187
A.4 The 13/7 Integer Discrete Wavelet Transform . . . 187
A.5 The CDF-9/7 Discrete Wavelet Transform . . . 188

B Arbitrary image dimensions **191**

C Coefficient scaling **197**

D Publications **199**
D.1 Accepted in a Journal 199
D.2 Submitted to Journal 199
D.3 IEEE Conferences and Others 200

List of Figures

1.1 MERBOTS Envisioned Scenario . . . 3
1.2 *Girona500* in *UWSim* reaching a target position requested by the operator 5
1.3 1200×650 image compressed in real time to 3047 bytes using DEBT and sent through a RF channel to the operator base in the surface . . . 6
1.4 Merbots GUI for underwater intervention missions . . . 7

3.1 Lifting: Predict and Update (forward and reverse) . . . 31
3.2 Normalized Processor(SPEC)×Memory(Latency^{-1}) performance gap . . . 33

4.1 Binary Entropy $\mathcal{H}(p)$ . . . 66
4.2 Reciprocal Significant bits/BIT . . . 67
4.3 Exponential Power Distribution ($\sigma = 1$) . . . 70
4.4 EPD η . . . 71
4.5 EPD $\Delta\mathbb{D}$ . . . 71
4.6 EPD $\Delta\mathcal{D}$ . . . 71
4.7 EPD η_1 . . . 72
4.8 EPD η_2 . . . 72
4.9 EPD η_3 . . . 72
4.10 EPD $\Delta\mathbb{D}_1$ . . . 74
4.11 EPD $\Delta\mathcal{D}_1$ . . . 74
4.12 EPD $\Delta\mathbb{D}_2$ . . . 74
4.13 EPD $\Delta\mathcal{D}_2$ . . . 74
4.14 EPD $\Delta\mathbb{D}_3$ . . . 74
4.15 EPD $\Delta\mathcal{D}_3$ . . . 74
4.16 1$^{\text{st}}$ refinement level distortion reduction ratio . . . 75
4.17 2$^{\text{nd}}$ refinement level distortion reduction ratio . . . 75
4.18 3$^{\text{rd}}$ refinement level distortion reduction ratio . . . 76
4.19 4$^{\text{th}}$ refinement level distortion reduction ratio . . . 76
4.20 5$^{\text{th}}$ refinement level distortion reduction ratio . . . 77
4.21 Reciprocal EPD $r_v(s)$ and $r_k(s)$ . . . 81
4.22 Lena (512 × 512) . . . 82

5.1 Orthogonal DWT band scanning order . . . 88
5.2 DEBT block diagram . . . 89
5.3 1 level transformation . . . 90
5.4 3 level transformation . . . 91
5.5 Maximum depth . . . 92
5.6 3 level transformation sets . . . 93

5.7 Tree partition . . . 96
5.8 Tree decomposition . . . 96
5.9 Set Scan Matrix . . . 98
5.10 Matrix of Sets . . . 99
5.11 Matrix of Blocks . . . 101
5.12 16×16 Lena . . . 107

6.1 Maxshift ROI . . . 120
6.2 Scaling ROI . . . 122
6.3 Nonlinear Scaling ROI . . . 122
6.4 Exclusive ROI . . . 123
6.5 Bitplane ROI . . . 123
6.6 Top Shift ROI . . . 124
6.7 Lena with ROI . . . 125
6.8 Lena ROI subband map . . . 126
6.9 Bitplane (left) and Scale (right) Lena ROI masks . . . 126
6.10 Bit, Map, and Scale Lena ROI at 250, 500, 1000, and 2000 bytes . . . 127

7.1 CDF-9/7 ns/pixel×pixel: paraline, lnaive and libdwt (deinterleaved) . . . 134
7.2 Airplane . . . 137
7.3 Airplane rate-distortion curve ($\bar{s} = 0.35$, $\bar{\sigma} = 7.66083$) . . . 137
7.4 Airplane rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 138
7.5 Low bit rate zoom for the airplane rate-distortion curve . . . 138
7.6 Baboon . . . 139
7.7 Baboon rate-distortion curve ($\bar{s} = 0.7$, $\bar{\sigma} = 17.4481$) . . . 139
7.8 Baboon rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 140
7.9 Low bit rate zoom for the baboon rate-distortion curve . . . 140
7.10 Barbara . . . 141
7.11 Barbara rate-distortion curve ($\bar{s} = 0.45$, $\bar{\sigma} = 16$) . . . 141
7.12 Barbara rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 142
7.13 Low bit rate zoom for the barbara rate-distortion curve . . . 142
7.14 Lena . . . 143
7.15 Lena rate-distortion curve ($\bar{s} = 0.45$, $\bar{\sigma} = 7.02501$) . . . 143
7.16 Lena rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 144
7.17 Low bit rate zoom for the lena rate-distortion curve . . . 144
7.18 Peppers . . . 145
7.19 Peppers rate-distortion curve ($\bar{s} = 0.45$, $\bar{\sigma} = 7.33603$) . . . 145
7.20 Peppers rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 146

7.21 Low bit rate zoom for the peppers rate-distortion curve . . . 146
7.22 Sailboat . . . 147
7.23 Sailboat rate-distortion curve ($\bar{s} = 0.6$, $\bar{\sigma} = 9.93486$) . . . 147
7.24 Sailboat rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 148
7.25 Low bit rate zoom for the sailboat rate-distortion curve . . . 148
7.26 Bridge . . . 151
7.27 Bridge rate-distortion curve ($\bar{s} = 0.45$, $\bar{\sigma} = 5.41702$) . . . 151
7.28 Bridge rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 152
7.29 Low bit rate zoom for the bridge rate-distortion curve . . . 152
7.30 Cathedral . . . 153
7.31 Cathedral rate-distortion curve ($\bar{s} = 0.35$, $\bar{\sigma} = 4.36203$) . . . 153
7.32 Cathedral rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 154
7.33 Low bit rate zoom for the cathedral rate-distortion curve . . . 154
7.34 Moon . . . 155
7.35 Moon rate-distortion curve ($\bar{s} = 0.25$, $\bar{\sigma} = 5.65685$) . . . 155
7.36 Moon rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 156
7.37 Low bit rate zoom for the moon rate-distortion curve . . . 156
7.38 Tree . . . 157
7.39 Tree rate-distortion curve ($\bar{s} = 0.4$, $\bar{\sigma} = 4.36203$) . . . 157
7.40 Tree rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green) . . . 158
7.41 Low bit rate zoom for the tree rate-distortion curve . . . 158
7.42 TasCPC shape (from variance and kurtosis) . . . 159
7.43 TasCPC images for rates 0.0001 and 0.0002 . . . 160
7.44 TasCPC images for rates 0.0005 and 0.001 . . . 161
7.45 TasCPC images for rates 0.005 and 0.01 . . . 161
7.46 Underwater images 16, 227, 274, 813, 977, 1082, and 1219 compressed at original, 500, 1000, and 2000 bytes . . . 162
7.47 Effect of ROI in PSNR: without ROI in black and with ROI in red . . . 166
7.48 Comparison of compressed image with and without ROI - DEBT . . . 167

B.1 1 dim transf. (ceil) . . . 192
B.2 1 dim transf. (floor) . . . 192

List of Tables

1.1 Image compression and transmission for underwater robotics 8
1.2 Comparison between the proposed and other compression algorithms . . . 11

3.1 Intel stream-inlift benchmark results (GB/s) 48
3.2 Intel lnaive ibior5x3 bandwidth saturation index - S_l 49
3.3 Intel paraline ibior5x3 bandwidth saturation index - S_p 49
3.4 Intel performane counters - ibior5x3 - 8 threads - 7680×4320 - 100 reps . 50
3.5 RPi3B stream-inlift benchmark results (GB/s) 52
3.6 RPi3B lnaive time (μs) 54
3.7 RPi3B paraline time (μs) 54
3.8 RPi3B Performane counters - ibior5x3 - 4 threads - 3840×2160 - 100 reps 55
3.9 RPi3B lnaive cdf9x7 (9 MOPs) time (μs) 56
3.10 RPi3B paraline cdf9x7 (2 MOPs) time (μs) 56

4.1 Distortion Reduction values for 9 bitplanes 61
4.2 Uniform/Laplace Distortion Reduction values for 9 bitplanes 62
4.3 Uniform root distortion reduction per bit ($\sqrt{3\sigma^2} = 2^9$) 78
4.4 Laplace root distortion reduction per sig bit and ref BIT ($\sigma^2 = 128^2$) . . . 80
4.5 Lena AC coeffs ($6 \times$ CDF-9/7 $\langle\sqrt{2}, \sqrt{2}\rangle$, $s = 0.15$, $\sigma = 32$) 83
4.6 Lena AC coeffs per bitplane ($6 \times$ CDF-9/7 $\langle\sqrt{2}, \sqrt{2}\rangle$, $s = 0.15$, $\sigma = 32$) . 83
4.7 Lena AC coeffs ($6 \times$ CDF-9/7 $\langle 1, 2\rangle$, $s = 0.45$, $\sigma \approx 7.025$) 84
4.8 Lena AC coeffs per bitplane ($6 \times$ CDF-9/7 $\langle 1, 2\rangle$, $s = 0.45$, $\sigma \approx 7.025$) . . 84

5.1 16×16 Lena pixel values (mean = 128) 108
5.2 16×16 Lena ibior13x7 DWT values 108
5.3 16×16 Lena coding 109

6.1 ROI PSNR comparison (db) 128

7.1 RPi2B stream-inlift benchmark results (GB/s) 130
7.2 RPi2B lnaive time (μs) 131
7.3 RPi2B paraline time (μs) 131
7.4 RPi2 lnaive cdf9x7 (9 MOPs) time (μs) 132
7.5 RPi2 paraline cdf9x7 (2 MOPs) time (μs) 132
7.6 Bandwidth Saturation Index - S_p - CDF-9/7 133
7.7 Airplane PSNR 149
7.8 Baboon PSNR 149
7.9 Barbara PSNR 149
7.10 Lena PSNR 149

7.11 Peppers PSNR 149
7.12 Sailboat PSNR 149
7.13 DEBT and JPEG-2000 timings (1258 images) - ratio 0.001 163
7.14 DEBT lossless timings (1258 images) 164
7.15 RPi2B and RPi3B timings (without ROI) 168
7.16 RPi2B and RPi3B timings (with ROI) 169
7.17 DEBT × JPEG-2000 170

A.1 Weights for 5/3 Integer DWT [2,2] $\langle 1, 2 \rangle$ 186
A.2 Weights for 9/3 Integer DWT [2,4] $\langle 1, 2 \rangle$ 187
A.3 Weights for 9/7 Integer DWT [4,2] $\langle 1, 2 \rangle$ 188
A.4 Weights for 13/7 Integer DWT [4,4] $\langle 1, 2 \rangle$ 189
A.5 Weights for CDF-9/7 DWT [4,4] $\langle 1, 2 \rangle$ 190

Abbreviations

ASC	**A**utonomous **S**urface **C**raft
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
ASWDR	**A**daptively **S**canned **W**avelet **D**ifference **R**eduction
AUV	**A**utnomouos **U**nderwater **V**ehicle
BIT	**B**inary dig**IT**
CABAC	**C**ontext **A**daptive **B**inary **A**rithmetic **C**oding
CDF	**C**ohen **D**aubechies **F**eauveau
CPU	**C**entral **P**rocessing **U**nit
DCT	**D**iscrete **C**osine **T**ransform
DEBT	**D**epth **E**mbedded **B**lock **T**ree
DWT	**D**iscrete **W**avelet **T**ransform
EPD	**E**xponential **P**ower **D**istribution
EZW	**E**mbedded **Z**eroTree **W**avelet
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
GPU	**G**raphics **P**rocessing **U**nit
GUI	**G**raphical **U**ser **I**nterface
HROV	**H**ybrid **R**emotely **O**perated **V**ehicle
JPEG	**J**oint **P**hotographic **E**xperts **G**roup
JPEG-2000	**J**oint **P**hotographic **E**xperts **G**roup **2000**
LSB	**L**east **S**ignificant **B**it
MOP	**M**emory **OP**eration
MSB	**M**ost **S**ignificant **B**it
MSE	**M**ean **S**quare **E**rror
PDF	**P**robability **D**ensity **F**unction
PSNR	**P**eak **S**ignal to **N**oise **R**atio

RD	**R**ate **D**istortion
RF	**R**adio **F**requency
ROI	**R**egion **O**f **I**nterest
SBC	**S**ingle **B**oard **C**omputer
SFF	**S**mall **F**orm **F**actor
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SPEC	**S**tandard **P**erformance **E**valuation **C**orporation
SPECK	**S**et **P**artitioning **E**mbedded blo**CK**
SPIHT	**S**et **P**artitioning **I**n **H**ierarchical **T**rees
UWSim	**U**nder**W**ater **S**imulator
VLSI	**V**ery **L**arge **S**cale **I**ntegration
WDR	**W**avelet **D**ifference **R**eduction

Chapter 1

Introduction

Initially, motivation, contextualization, and problem statement, associated with the state-of-the-art in underwater applications with real-time image compression and the limitations of current algorithms is presented, followed by the presentation of the main contributions of this thesis. Finally, a summary and outline of this document is presented.

1.1 Motivation

Robotic applications and, particularly, Autonomous Underwater Vehicles for Intervention (I-AUV) use images from its built-in camera(s) as one of its main sources of data, among others, in order to control its internal algorithms. In a supervised system, these images should reach the operator with the lowest latency and with the highest quality possible so that he can interact with the system and adjust the task execution in a supervised manner.

Besides this, communications is a crucial subsystem in any robotic application, specially the ones that permit the user to interact remotely with the system. Because of that, image compression and transmission is necessary in order to send the required information with the lowest latency and without compromising the network and the whole system.

According to Arrichiello et al. [1] aquatic acoustic channels suffer from significant frequency and distance dependent attenuation, extensive time-varying multipath, motion-induced Doppler distortion and extreme channel latency due to the low speed of sound.

Therefore, the available bandwidth is strongly limited and distance dependent. Due to all these limitations, the introduction of intra-vehicle exchanged information in the control loop of marine robots can degrade the overall system performance. Deployment of underwater robot teams is challenging due to both issues concerning motion control as well as the development and use of a communication infrastructure.

Existing image compression algorithms, in most part, have not been optimized for such low bit rates and perform sub optimally in this scenario. Also, the variable dynamic nature of the communications channel's available bandwidth requires the flexibility of using dynamically varying packet sizes. In summary, instead of having the data (image) dictate how many bytes to transmit, it would be better to adapt the data to the available amount of bytes that can be transmitted at any point in time.

In addition, image compression algorithms are usually designed for general purpose applications and not specifically for real time robot control under constrained communication channels. Most design decisions that guide their formulation naturally give high priority to compression ratio and ignore other factors which are relevant in more limited scenarios, which may restrict both computing power and energy consumption, and with communications channels that present high delay and low bandwidth.

One more point which differentiates underwater applications is the comparatively high cost per frame, i.e., it is usually quite expensive to set up an underwater intervention and high quality images, preferably exact copies of what the image sensor has provided, should be stored locally to be later analyzed and archived. In addition, the on board computing facilities are usually limited to small battery operated low power computers, e.g., ARM based Single Board Computers (SBC), or small form factor (SFF) desktop class computers.

1.2 Context

Upfront, it should be made clear that the current work deals exclusively with the compression of images and does not actually touch on the subject of transmission and its protocols, including error detecting and correcting protocols. In fact, the context described in this section served as the main motivation for a general purpose compression algorithm with all the required features that were gathered from previous attempts at

image transmission in underwater scenarios, including transmission errors. The characterization of underwater images described in section 7.4 shows that these images possess the same statistics as natural images, which can be efficiently modeled by a known Probability Density Function (PDF). Therefore, a solution to the problem of underwater image compression for extremely low bandwidth scenarios is also applicable to more general scenarios and has a broader applicability, beyond the boundaries of underwater scenarios.

This thesis is associated the MERBOTS project, which requires the use of wireless communications to control a robot, if necessary, without using an umbilical as seen in Figure 1.1. Several experiments have been performed in this area, using both sonar and Radio Frequency (RF) modems [2, 3].

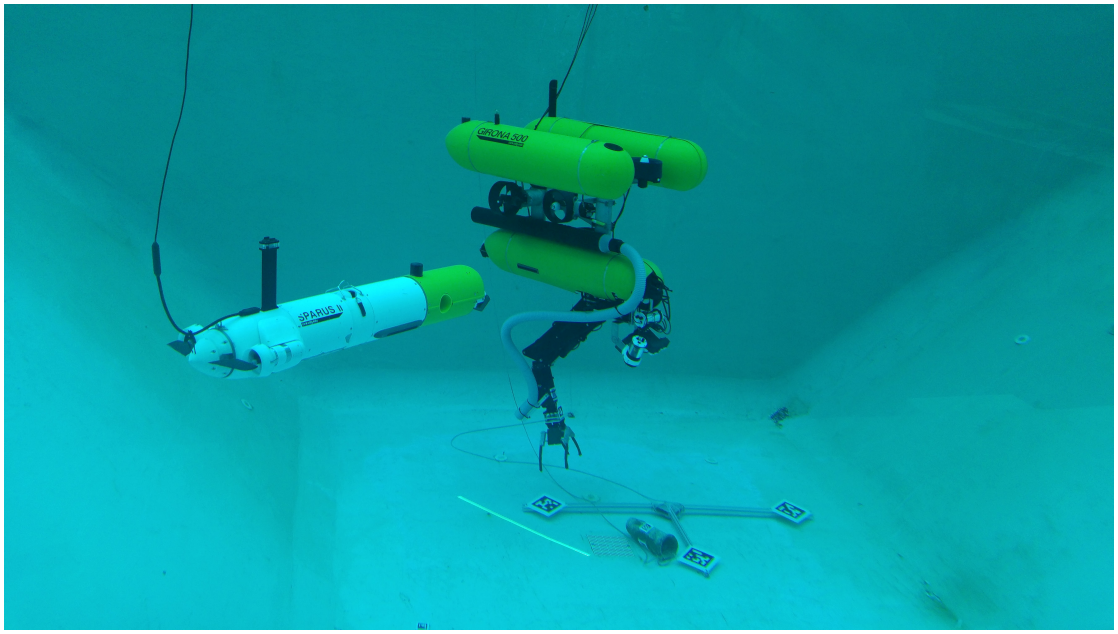


FIGURE 1.1: MERBOTS Envisioned Scenario

MERBOTS represents a natural evolution of previous research projects in the field of underwater robotic intervention (e.g. RAUVI [4], TRITON [5], EU FP7 TRIDENT [6]) and one of its objectives is to provide different communications technologies to allow the operation of a vehicle without any physical connection to the surface operators, which are supervising and controlling an intervention task.

The MERBOTS (DPI2014-57746-C3-1-R) coordinated project has been organized into 3 subprojects:

- MERMANIP (DPI2014-57746-C3-1-R), under responsibility of UJI (Jaume I University), in charge of the multi-sensory based autonomous manipulation, the multimodal user interface, and the Sonar/RF communication system for enabling compressed image transmissions between the robots and the human operator.
- ARCHROV (DPI2014-57746-C3-3-R), under responsibility of UdG (University of Girona), assuming the cooperative mobile robotics part, including communications and localization of the mobile robots, sonar-based survey, and path planning, also the construction of a new ASC and the final mechatronics, hardware/software integration will be under their responsibility.
- SUPERION (DPI2014-57746-C3-2-R), under responsibility of UIB (University of Balearic Island), in charge of processing the multi-modal sensor data collected during the survey stage to build 3D models of the area of intervention and the target, as well as for searching the target prior to the intervention and tracking the target during the intervention.

Under short range conditions, acoustic modems are capable of transmitting data at a maximum rate of tens of thousands of bits per second and, for even shorter distances, RF (Radio Frequency) modems can achieve similar speeds [7]. Realistically, the expected bandwidth is in the range of a few hundred to a few thousand bits per second with high variance. Other difficulties include high error rates and packet loss, which must be dealt by using appropriate communications protocols.

At such low bit rates, however, even the best possible image compression algorithm will not be able to convey a detailed representation of the captured image and, in this case, only a reduction of the source information would help. This reduction can be done by selecting small areas of the image (foreground) which are then amplified in such a way that they become more important than they really are and are partially coded before the rest of the image (background), effectively blending them in such a way that results in a more detailed foreground in exchange of a less detailed background while keeping both and allowing for some context information to be provided.

It is also important to be able to analyze the captured images when the underwater mission is over, when the data can be collected directly from the Autonomous Underwater Vehicle (AUV) storage, and that these images are identical to the captured ones, without any artifacts resulting from its compression (lossless compression).

The work presented here contributes for the implementation of an efficient communications protocol which is an ongoing effort and deals with the transmissions of commands, telemetry, and images, among other data, in a unified way. Initial transmission tests in the context of the Merbots project using the results of this thesis is presented in [2] which uses a bidirectional transport protocol for acoustic wireless teleoperation of an underwater robot using high level commands and the Depth Embedded Block Tree (DEBT) progressive image compression algorithm. Figure 1.2 shows the Girona 500 AUV in the UnderWater Simulator (UWSim) [8] where some compression parameters can be specified.

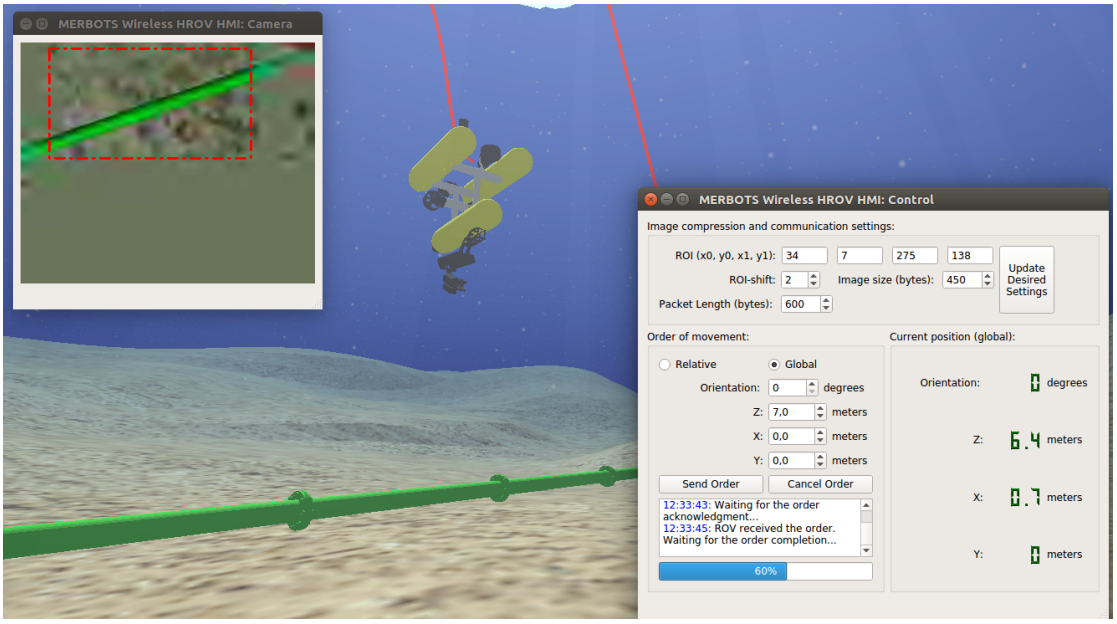


FIGURE 1.2: *Girona500* in *UWSim* reaching a target position requested by the operator

Actual underwater transmission tests have already been performed by the Merbots group at the University Jaume I (UJI) using the Interactive and Robotic Systems Lab (IRS-Lab) robot using various compression ratios with promising results. The flexibility derived from the variable compressed image size, coupled with low distortion at high compression ratios, greatly facilitates the communications protocol design by allowing for the fast adaptability under varying latency and bandwidth conditions [2]. Figure 1.3 shows a color image received by the surface operator which was compressed in real time

and transmitted through a RF channel using the proposed algorithm presented in this thesis called Depth Embedded Block Tree (DEBT).

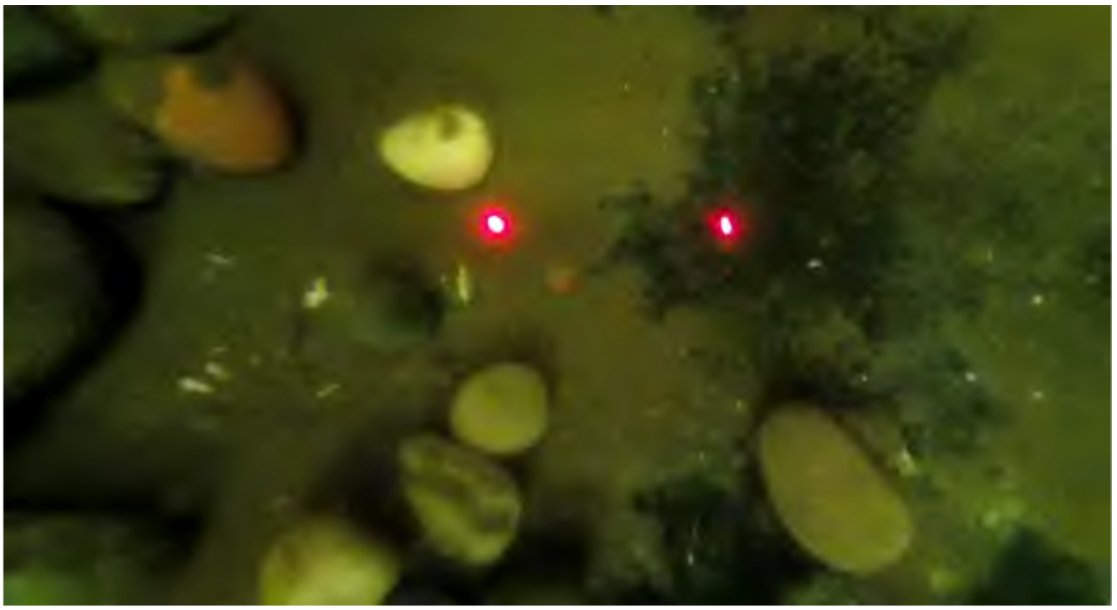


FIGURE 1.3: 1200×650 image compressed in real time to 3047 bytes using DEBT and sent through a RF channel to the operator base in the surface

In summary, MERBOTS aims to bring a team of heterogeneous marine robots (an Autonomous Surface Craft (ASC), an AUV and a Heterogenous Remotely Operated Vehicle (HROV)) together, tightly cooperating to conduct a multimodal survey (stereo, laser and multibeam) of an unknown, unstructured area with significant 3D relieve (like a shipwreck), where a multifunctional intervention operation must be performed. A mixture of autonomous (ASC and AUV) and task-level teleoperated vehicles (HROV) has been developed. So, MERBOTS has been able to integrate recent and promising technologies to explore the powerful concept of a wireless HROV, in the cutting-edge of technology. Figure 1.4 shows the Merbots Graphical User Interface (GUI) for underwater intervention missions.

1.3 Available Solutions

Most available solutions try to adapt their communications protocols to an available image compression algorithm/implementation. This way, the chosen algorithm serves as both a facilitator, by supplying the compressed image, but also as a constrainer, by imposing its limitations on the design of the protocol. Variable low bandwidth does not

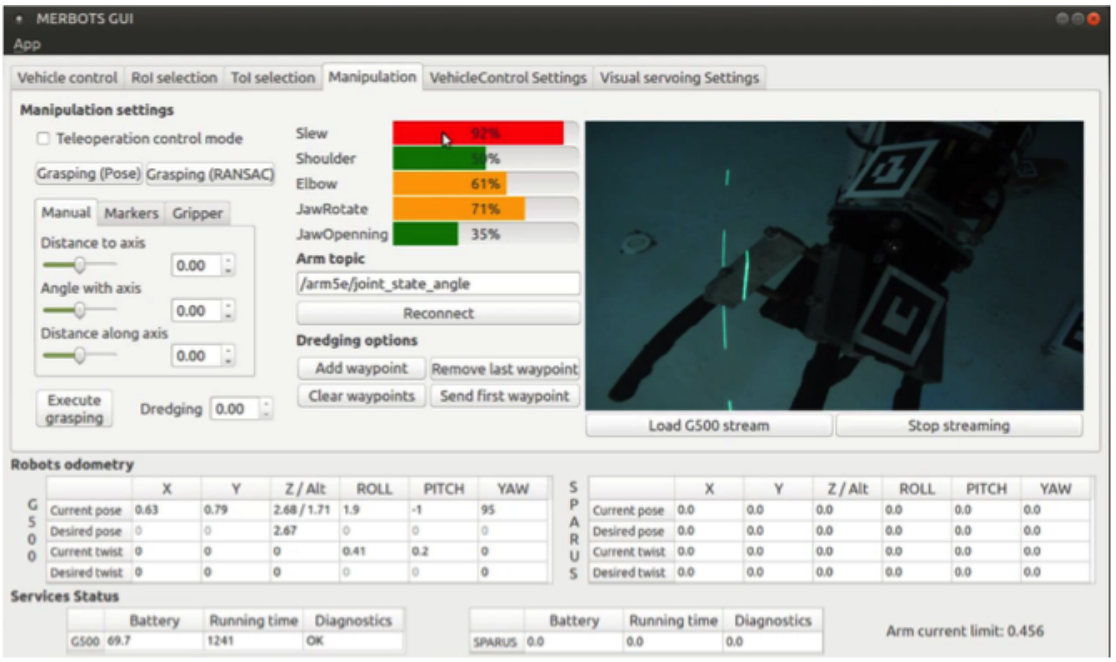


FIGURE 1.4: Merbots GUI for underwater intervention missions

cope well with the requirement to transmit fixed amounts of data and this presents a major hurdle in effective and timely communications. Flexibility and compression ratio are usually inversely related and the exclusive focus on compression ratio used by most compression algorithms usually makes them unsuitable for this particular communications scenario.

The main bibliographical references associated with image compression and transmission for underwater robotics are summarized in Table 1.1, which presents the main features of the referenced works detailed in chapter 2. This table is not suited for comparing the cited works but only serves as a reminder of each work's main features for future reference.

None of the references presented in Table 1.1 actually solve the problem of underwater image communications partly because most of them employ off-the-shelf general compression algorithms to tackle a very specific task, which requires the simultaneous use of many techniques which might be of no importance in the general case and were not taken into consideration in the original design of the algorithm.

According to Kaeli [22] in his MIT PhD thesis, the current standard state-of-the-art image compression algorithm Joint Photographic Experts Group 2000 (JPEG-2000) [23]

TABLE 1.1: Image compression and transmission for underwater robotics

Solution	Feature 1	Feature 2
Suzuki [9]	DCT and JPEG	16 kbps
ASIMOV [10]	2 Frames/s	30 kps
Japan [9]	MPEG4	10Frames/s
Hong [11]	DWT	30Frames/s
Khamene [12]	DWT	128x128 pixel frames
Konstantinos [13]	DCT	150 kbps
Eastwwod [14]	DCT and JPEG	EPIC
Walter [15]	WDR/DWT	ROI
Pearlman [16]	DWT	Low complexity
Murphy [17]	Telemetry	Acoustic channel
Zheng [18]	Telemetry	Acoustic channel
Senapati [19]	WBTC	DCT/DWT
Zhang [20]	Hybrid wavelets	Directional filter banks
Mohammed [21]	SPIHT	Reducing PAPR
Kaeli [22]	JPEG-2000	Larger packets

employs variable compression rates using progressive encoding, meaning that a compressed image can be transmitted in pieces or packets that independently add finer detail to the received image. This is particularly well suited to underwater applications where acoustic channels are noisy and subject to high packet loss, however, JPEG-2000 [23] is optimized for larger packets that are unrealistic for underwater acoustic or radio frequency (RF) transmissions [22].

Kaeli [22] also compares Joint Photographic Experts Group (JPEG) [24], Joint Photographic Experts Group 2000 (JPEG-2000) [23] and Set Partitioning in Hierarchical Trees (SPIHT) [25]. JPEG [24] is a common example of a lossy compression format which uses the DCT (Discrete Cosine Transform) for each 8x8 block to achieve roughly 10:1 compression without major perceptual changes in the image. Recent work [22] has focused on using similar wavelet decomposition techniques for underwater applications using smaller packet sizes with the SPIHT [25] image compression algorithm, which generates a bit-oriented output stream and allows for its truncation at any point (exact output size). SPIHT [25], however, lacks many features available with JPEG-2000 [23] like a lossless mode and Region Of Interest (ROI) coding.

Other techniques [17] involve highly specialized scene recognition algorithms which are error prone and not general in nature which makes their application very specific.

However, in his seminal paper "Embedded Image Coding Using Zerotrees of Wavelet Coefficients" [26], Jerome M. Shapiro says:

> Wavelet techniques show promise at extremely low bit rates because trends, anomalies, and information at all "scales" in between are available. A major difficulty is that fine detail coefficients representing possible anomalies constitute the largest number of coefficients, and therefore, to make effective use of the multiresolution representation, much of the information is contained in representing the position of those few coefficients corresponding to significant anomalies.

which seems to indicate that the use of multi-resolution decomposition coupled with a good representation for the significance map should yield good results for extremely low bit rates. In fact, Shapiro goes on:

> An important aspect of low bit rate coding is the coding of the positions of the significant coefficients (significance map). In order to achieve very low bit rates using scalar quantization followed by entropy coding, the probability of a 0 (non-significance) must be extremely high. Typically, a large fraction of the bit budget is spent on the encoding of the significance map, or the binary decision as to weather a sample, in this case a coefficient of a 2-D DWT (Discrete Wavelet Transform), has zero or nonzero quantized value.

which makes it clear that the coding of the significance map is of paramount importance in order to achieve good low bit rate compression, and goes on to describe the grouping of coefficients in zerotrees and their encoding. Nevertheless, the effectiveness of the EZW (Embedded Zerotree Wavelet) [26] algorithm is highly dependent on the othogonality or near-orthogonality of the transform used in the sense that it should preserve the energy across the subbands coupled with the depth-first nature of the tree search (partitioning) data structure.

1.4 Proposed Solution

This thesis focuses on solving the problem detected by Kaeli [22] by designing a new fast progressive image compression parallel algorithm with ROI which outputs an embedded

rate-distortion optimized stream and addresses very low bit rate compression with user defined packet sizes suitable for the implementation of any communications protocols, either underwater or in any other scenario, while remaining competitive with current state-of-the-art compressors [16, 23, 25–29] at other bit rates.

Embedded image compression is a very efficient way to cope with varying transmission bandwidth problems in hard real time systems, where it would be better to have a low quality version of the current image instead of a high quality version of an old image. Also, in a totally different context, there is no need to transmit a high resolution image when the user will be visualizing it in low resolution, i.e., better use of the available bandwidth can be achieved by transmitting an image which is closer to the viewer resolution instead of the resolution it was originally compressed with.

While most embedded compression algorithms are designed to be quality scalable (any prefix of the bitstream yields the "best" lower quality version of the original), resolution scalability or a combination of quality and resolution scalability is a desired property and, in many cases, yields a "better" subjective version of the original image. Also, color channel scalability, where the luminance (luma) channel is encoded separately from the other color channels (chroma), could also be used to further decrease the minimum amount of data necessary for a first "acceptable" version of the original image, minimizing the overall delay between image acquisition and display at a remote location.

In addition to the above scalability properties, the addition of Region Of Interest (ROI), where parts of the image are dealt with higher priority than others allowing these parts to take precedence and reach the destination sooner and with more quality than the other parts, is a desirable property. This would impose a negative impact on the final compression ratio of the whole image but would partition the image into areas that are more "important" than others, allowing for better use of the available bit budget, reserving more bits to these "important" areas.

Embedded image compression algorithms based on multi-resolution wavelet transforms, grouping of similar magnitude coefficients into sets, and bitplane scanning order have proven to be quite successful, usually with a very low complexity over compression ratio. Even though these methods are somewhat rigid and do not provide a clear algorithmic separation between the modeling and coding, they are usually very fast and present very good compression results, even without a final entropy coding of the resulting stream.

Table 1.2 lists the main requirements of the image compression subsystem and also compares the Depth Embedded Block Tree (DEBT) algorithm [7], developed in this thesis, with the Joint Photographic Experts Group (JPEG) [24], Joint Photographic Experts Group 2000 (JPEG-2000) [23], and Set Partitioning in Hierarchical Trees (SPIHT) [25] standards. JPEG [24] is an aging algorithm that, although being quite fast, performs very poorly under high compression (it was never designed for this purpose) and does not possess the necessary features. JPEG-2000 [23], on the other hand, is a quite complex and sophisticated algorithm that compresses well under almost all conditions and has most of the needed features but is quite slow and also does not perform well under very low bit rates scenarios. SPIHT [25] is a set-partitioning progressive algorithm which performs extremely well for natural images but has restrictions on the possible transforms that can be used with it and does not perform very well for extremely low bit rates.

TABLE 1.2: Comparison between the proposed and other compression algorithms

	DEBT	JPEG	JPEG-2000	SPIHT
Quality progressive compression	✓	✗	✓	✓
Resolution progressive compression	✓	✗	✓	✗
Region Of Interest (ROI)	✓	✗	✓	✗
Nonlinear Scaling Region Of Interest (ROI)	✓	✗	✗	✗
Rate Distortion Optimized	✓	✗	✓	✗
Real time (fast) compression	✓	✓	✗	✓
Exact size (truncate) compression	✓	✗	✗	✓
Bit oriented progressive stream	✓	✗	✗	✓
Lossless and Lossy compression	✓	✗	✓	✗
Parallel algorithm	✓	✗	✗	✗
Applicable for underwater acoustic or RF transmissions	✓	✗	✗	✓

Parallelism was a primary concern during all phases of the algorithm design and played a major role in the decisions, both major and minor, regarding the trade offs between compression ratio and performance.

The solutions developed in this thesis are applied in research projects associated with underwater robotics developed at the University Jaume I (UJI) in partnership with the University of Girona (UdG). In the context of the MERBOTS research project (`http://www.irs.uji.es/merbots`), a three-year coordinated project funded by the Spanish government for the period 2015-2017 under grant DPI2014-57746-C3 [30], one of the objectives is to build a wireless communication system that can provide freedom of movements to the underwater robot and, at the same time, allow the operator to get

feedback and supervise the intervention (Figure 1.1). The robotic system development assists the archaeologists in the detailed work of monitoring, characterization, study, reconstruction and preservation of archaeological sites, always in accordance with the continuous supervision of a human expert.

1.5 Contributions

The main contributions of this thesis are:

1. Transformation latency minimization: The transformation step is usually responsible for a significant part of the total execution time of any transform-based compression algorithm. The DWT (Discrete Wavelet Transform) is perhaps the most used transform for contemporary algorithms and an efficient implementation would benefit all current DWT-based encoders. An optimal, minimal time parallel algorithm is presented and it is shown that the DWT can be performed in the same time as a simple inplace memory copy, i.e., given that all coefficients necessarily have to be read and written once, the presented algorithm achieves minimal running time;

2. Compressed output bit stream prioritization: Transform coefficients are modelled by a Probability Density Function (PDF) which is used to calculate the average distortion reduction per bit (entropy) for significance and refinement bits of coefficients in a certain range of magnitudes, for all ranges. Sorting all these values in decreasing order, properly weighted by the DWT subband gains, gives the best order to scan the coefficients in such a way as to minimize the distortion at every step, on average. It is shown that, under certain circumstances, some refinement bits of higher range coefficients should be encoded before the significant bits of lower range coefficients for the same bitplane;

3. Fast progressive set partitioning image compression algorithm: The zerotree and the zeroblock concepts are extended to allow for the dynamic ordering of significant and refinement bits while efficiently coding of the significance map, which is specially important at low bit rates. The Depth Embedded Block Tree (DEBT) algorithm is progressive, produces a quality or resolution scalable embedded bitstream which can be truncated at any point yielding the “best” representation

of the original image for the resulting prefix, supports the use of any DWT with arbitrary subband gains, and is lossless when using DWTs that map integers into integers [31];

4. General Region Of Interest (ROI) coding with non-linear scaling: by delaying the encoding of the lower foreground bitplanes which consist of a large number of mostly noisy data, it is possible to improve the context information (background) by a substantial margin. The DEBT algorithm can also efficiently cope with complex ROI geometries (which cannot be compactly described and would be prohibitively expensive to be sent as overhead information) by using exclusive ROI masks and not sending any map information at all at the cost of slightly worse reconstruction quality;

5. Available implementation: The DEBT algorithm is implemented for an ARM SBC platform for underwater applications with bandwidth restriction and provides lossless rate-distortion optimized progressive embedded encoding with non-linear ROI. It also makes use of the minimum-time parallel DWT algorithm and does not use any entropy coding step in order to achieve fast run times, which in fact solves the problems detected in the recent literature (Kaeli [22]).

1.6 Organization

This thesis is structured as follows:

1. Chapter 1 introduces the motivation, contextualizes and describes the problem, presents the main available solutions in underwater image communications, describes the proposed solution, and lists the contributions of this thesis;

2. Chapter 2 describes related work on underwater image compression, image compression in general and parallelization schemes for the DWT.

3. Chapter 3 presents the paraline minimal time DWT algorithm together with extensive analysis from the memory saturation point of view and running time performance on multicore ARM SBC and Intel Desktop class computers;

4. Chapter 4 derives the best ordering, according to the squared error distortion measure, in which to encode groups of coefficients by means of a list sorted in decreasing order of distortion reduction. This list is independently derived by both encoder and decoder so that it does not need to be sent as side information;

5. Chapter 5 uses the results from Chapters 3 and 4 and the DWT subband gains (Appendix A) to describe in detail the DEBT algorithm and the concepts of variable depth zerotrees and zeroblocks which are used to compactly encode the significance map;

6. Chapter 6 describes general ROI concepts, classifies the type of ROI based on the need to encode the ROI map, introduces non-linear ROI scaling, and compares the many different ROI encoding schemes;

7. Chapter 7 presents results achieved from using available sets of underwater imagery and also from real underwater experiments;

8. Chapter 8 states the conclusion and points some directions for future work which were identified;

9. Appendix A contains the subband weight tables for various DWT;

10. Appendix B derives the parent-child set relationships for arbitrary image dimensions;

11. Appendix C derives the relations for reversible coefficient scaling;

12. Appendix D lists the publications produced during the research.

Chapter 2

Related Work

The following is a literature review associated with image compression and transmission and underwater robotics, as well as image compression in general and parallelization schemes for the Discrete Wavelet Transform (DWT) computation.

2.1 Image Compression and Transmission and Underwater Robotics

Suzuki and Sasaki [9] was the first system to demonstrate image transmission over a vertical path which was developed in Japan. The JPEG standard DCT (Discrete Cosine Transform) was used to encode 256x256 pixel still images with 2 bits per pixel. Transmission of about one frame per 10 seconds was achieved using 4-DPSK (Differential Phase Shift Keying) at 16 kbps. Remarkable results obtained with this system included a video of a slowly moving crab, transmitted acoustically from a 6,500 m deep ocean trench. Another vertical path image transmission system was developed in France and successfully tested in 2,000m deep water. This system was also based on the JPEG standard and used binary DPSK for transmission at 19 kbps.

An image transmission system has been developed in a Portuguese effort called ASIMOV [10]. In this project, a vertical transmission link is secured by a coordinated operation of an AUV and an ASC (Autonomous Surface Craft). Once the site is chosen and the vehicles are positioned, transmission of a sequence of still images of about 2 frames/sec is accomplished at 30 kbps using an 8PSK (Phase-shift keying) modulation method.

Another experiment of underwater video transmission system, developed in Japan [9], employs 4PSK, 8PSK, and 16QAM (Quadrature Amplitude Modulation) signals with 40 KHz bandwidth to achieve transmissions at up to 128 kbps. The system uses 100 kHz carrier frequency and was tested over a short vertical path of 30 m. The MPEG-4 standard was employed for video compression, and a frame rate of 10 frames/sec was supported. Efficient compression can be achieved if there is a-priori information available about the images to be taken. Algorithms that exploit the properties inherent specifically to underwater images are such an example.

Because underwater images have low contrast, their information is concentrated at low frequencies. Thus, by decomposing the image information into low and high frequency subbands, and encoding the low bands with more precision, it is possible to achieve higher compression ratios. This is the basic motivation behind the work in [11] which used the DWT (Discrete Wavelet Transform) in place of the standard DCT. This algorithm was applied to a sequence of underwater images, taken at 30 frames per second, each having 256x256 8-bit pixels. The achieved compression ratio of 100:1 provided very good quality monochrome video. The resulting bit rate needed to support such high quality is on the order of 160 kbps, which surpasses the capabilities of the current acoustic modem technology. The DWT is combined with entropy-constrained vector quantization (ECVQ) and motion-compensated prediction to achieve an average of 0.08 bits/pixel. However, the algorithm is equally applicable to reduced-size images. For example, a 144x176 pixel image would require 60 kbps with 30 frames per second, or 20 kbps with 10 frames per second. These values are approaching the capabilities of an acoustic modem, provided that a bandwidth-efficient modulation/detection scheme is used.

Another system that exploits wavelet based compression together with motion compensation is proposed in [12]. Although it attains approximately the same compression ratio (100:1) as in [11], it has better visual intelligibility because it employs a generalized dynamic image model (GDIM) that decouples the geometric and photometric variations in an image sequence commonly encountered in deep sea imagery. This approach is in contrast with ordinary terrestrial motion-compensated algorithms, where steady and uniform illumination is the underlying assumption. Using 128x128 pixel frames and 30 frames/sec, the resulting bit rates needed to support real-time video transmission were in the order of 40 kbps. The traditional methods described above fall into the category

of hybrid methods, because they combine image compression with motion compensation. A different approach is emerging in the form of model-based video compression methods.

Pelekanakis [13] presents a high bit rate acoustic link for underwater video transmission. Currently, encoding standards support video transmission at bit rates as low as 64 kbps. While this rate is still above the limit of commercially available acoustic modems, prototype acoustic modems based on phase coherent modulation/detection have demonstrated successful transmission at 30 kbps over a deep water channel. The key to bridging the remaining gap between the bit-rate needed for video transmission and that supported by the acoustic channel lies in two approaches: use of efficient image/video compression algorithms and use of high-level bandwidth-efficient modulation methods.

An experimental system [13], based on DCT and Huffman entropy coding for image compression, and variable rate Mary quadrature amplitude modulation (QAM) was implemented. Phase-coherent equalization is accomplished by joint operation of a decision feedback equalizer (DFE) and a second order phase locked loop (PLL). System performance is demonstrated experimentally, using a transmission rate of 25000 symbols/sec at a carrier frequency of 75 kHz over a 10 m vertical path. Excellent results were obtained, thus demonstrating bit rates as high as 150 kbps, which are sufficient for real-time transmission of compressed video. As an alternative to conventional QAM signaling, whose high-level constellations are sensitive to phase distortions induced by the channel, Mary differential amplitude and phase shift keying (DAPSK) was used. DAPSK does not require explicit carrier phase synchronization at the receiver, but instead relies on simple differentially coherent detection. Receiver processing includes a linear equalizer whose coefficients are adjusted using a modified linear least square (LMS) algorithm. Simulation results confirm good performance of the differentially coherent equalization scheme employed.

Eastwood et al. [14] presents techniques for compression of laser line scan and camera images, as well as format specific data compression for quick-look sonar mapping data. For image compression, both JPEG and a wavelet based technique called Efficient Pyramid Image Coder (EPIC) are examined. JPEG is found to be less efficient than the wavelet transform but has the advantage of being robust with respect to lost data packets. The wavelet based transform is more efficient at high compression rates though

above a certain rate both offer similar performance. The specific context for this work is the autonomous mine hunting and mapping technology (AMMT) demonstration which utilizes the DARPA large diameter underwater vehicle. The ultimate goal of the project is identification and imaging of mine-like objects. The algorithms presented here were implemented in real-time and operated in the field for this program. Details of specific issues encountered during development of the system are also described.

Walter et al. [15] presents a new wavelet-based image compression system. The compression system is based on a particular type of compressed encoding of wavelet transforms called Wavelet Difference Reduction (WDR) and describes experimental results in applying a compression algorithm to a suite of underwater camera images. These underwater camera images were required to be compressed at very high compression ratios (400:1, 200:1, 100:1, and 50:1) and the algorithm produced very high-fidelity approximations. In fact, it performed at a comparable level to a system based on the celebrated Daub CDF-9/7 system (used in JPEG-2000 [32]) yet employing 256 times less RAM (Random Access Memory) and a 16-bit dynamic range (with 8-bit images) instead of a 32-bit dynamic range. This system has the following advantages: low-power utilization, low RAM requirements, embedded bit-plane compression, scalable decompression, and region-of interest (ROI) selectivity. The underwater camera images were required to be compressed at very high compression ratios (400:1, 200:1, 100:1, and 50:1), and yet the algorithm produced very high-fidelity decompressions. They shall illustrate all of these advantages in an application to very high compressions of underwater camera images. The compression system is based on a particular type of compressed encoding of wavelet transforms called Wavelet Difference Reduction (WDR). The coding scheme enjoys the advantages of scalability and ROI selectivity.

Pearlman et al. [16] proposes an embedded, block-based, image wavelet transform coding algorithm of low complexity. It uses a recursive set partitioning procedure to sort subsets of wavelet coefficients by maximum magnitude with respect to thresholds that are integer powers of two. It exploits two fundamental characteristics of an image transform: the well defined hierarchical structure and energy clustering in frequency and in space. They describe the use of this coding algorithm in several implementations, including reversible (lossless) coding and its adaptation for color images, and show extensive comparisons with other state-of-the-art coders, such as SPIHT and JPEG2000. They conclude that this algorithm, in addition to being very flexible, retains all the desirable features of these

algorithms and is highly competitive to them in compression efficiency. The two partition strategies allow for versatile and efficient coding of several image transform structures, including dyadic, blocks inside subbands, wavelet packets, and discrete cosine transform (DCT).

Murphy [17] presents an analysis of the unique considerations facing telemetry systems for free-roaming Autonomous Underwater Vehicles used in exploration. These considerations include high-cost vehicle nodes with persistent storage and significant computation capabilities, combined with human surface operators monitoring each node. He then proposes mechanisms for interactive, progressive communications of data across multiple acoustic hops. These mechanisms include wavelet-based embedded coding methods, and a novel image compression scheme based on texture classification and synthesis. The specific characteristics of underwater communication channels, including high latency, intermittent communication, the lack of instantaneous end-to-end connectivity, and a broadcast medium were taken into consideration. Human feedback is incorporated by allowing operators to identify segments of data that warrant higher quality refinement, ensuring efficient use of limited throughput. Murphy then analyze the performance of these mechanisms relative to current practices. Finally, present CAPTURE, a telemetry architecture that builds on this analysis. CAPTURE draws on advances in compression and delay tolerant networking to enable progressive transmission of scientific data, including imagery.

Zheng et al. [18] presents a special application of delay tolerant networks (DTNs). Efficient data collection in deep sea poses some unique challenges, due to the need for timely data reporting and the delay of acoustic transmission in the ocean. Autonomous underwater vehicles are deployed in deep sea to surface communications and frequently have to transmit collected data from sensors (in a 2-dimensional or 3-dimensional search space) to the surface stations. However, additional delay occurs at each resurfacing. The paper want to minimize the average data reporting delay, through optimizing the number and locations of AUV resurfacing events.

Senapati et al. [19] presents a listless implementation of a wavelet based block tree coding (WBTC) algorithm of varying root block sizes. The WBTC algorithm improves the image compression performance of SPIHT at lower rates by efficiently encoding both inter and intra scale correlation using block trees. Though WBTC lowers the memory

requirement by using block trees compared to SPIHT, it makes use of three ordered auxiliary lists. The proposed algorithm is combined with DCT and DWT to show its superiority over DCT and DWT based embedded coders, including JPEG-2000 at lower rates. The compression performance on most of the standard test images is nearly the same as WBTC but it outperforms SPIHT by a wide margin particularly at lower bit rates. This feature makes {WBTC} undesirable for hardware implementation; as it needs a lot of memory management when the list nodes grow exponentially on each pass. The proposed listless implementation of {WBTC} algorithm uses special markers instead of lists. This reduces dynamic memory requirement by 88% with respect to {WBTC} and 89% with respect to SPIHT.

Zhang et al [20] presents a new underwater video compression technique based on adaptive hybrid wavelets and directional filter banks to achieve both high coding efficiency and good reconstruction quality at very low-bit rates. A key application is the real-time transmission of video through acoustic channels with limited bandwidth from an autonomous underwater vehicle to a surface station, e.g., for man-in-the-loop monitoring and inspection operations. For intra-frame coding, the method maintains details in texture regions at relatively low bit rates, and overcomes the ringing artifacts within smooth regions. For inter-frame coding, improved efficiency is achieved by making use of: (1) a new spatio-temporal just noticeable distortion model to remove perceptual redundancy; (2) motion interpolation to reduce bit rate; and (3) variable precision in quantizing the residual error. Experiments with underwater video sequences are presented to assess the effectiveness of the proposed approach, in comparison to traditional wavelet-based techniques.

According to Esmaiel [33, 34] the SPIHT coder based on the wavelet algorithm is probably the most widely used for image compression, as well as being a basic standard of compression for all subsequent algorithms [35–37]. In SPIHT, the information bits are sorted according to the bit information significance. The protection level of transmitted data must take this feature into account and progressive protection is provided to the transmitted bits. This methodology is used to reduce the distortion in the reconstructed image (reduce the difference between the original and the reconstructed images). After image decomposition with CDF-9/7 wavelet, the general SPIHT coding algorithm encodes images by splitting the decomposed image into considerable sections on the basis of the significance classification function [38].

Mohammed and Hamada [21] propose new scheme for efficient rate allocation in conjunction with reducing peak-to-average power ratio (PAPR) in orthogonal frequency-division multiplexing (OFDM) systems. Modification of the SPIHT image coder is proposed to generate four different groups of bit-streams relative to its significances. The significant bits, the sign bits, the set bits and the refinement bits are transmitted in four different groups. he proposed method for reducing the PAPR utilizes twice the unequal error protection (UEP) using the Read-Solomon codes (RS) in conjunction with bit-rate allocation. The output bit-stream from the source code (SPIHT) will be started by the most significant types of bits (first group of bits).

According to Kaeli [22] in his MIT PhD thesis, the current standard state-of-the-art image compression algorithm JPEG-2000 [23] employs variable compression rates using progressive encoding, meaning that a compressed image can be transmitted in pieces or packets that independently add finer detail to the received image. This is particularly well suited to underwater applications where acoustic channels are noisy and subject to high packet loss, however, JPEG-2000 [23] is optimized for larger packets that are unrealistic for underwater acoustic transmissions [22]. They hope to implement this framework on a physical embedded system aboard a vehicle.

While automated classification algorithms can lessen the burden on human annotators after a mission, most are too computationally expensive or lack the robustness to run in situ on a vehicle [22]. Fast algorithms designed for mission-time performance could lessen the latency of understanding by producing low-bandwidth semantic maps of the survey area that can then be telemetered back to operators during a mission. One realization is to operate directly off the image buffer in the sealed camera pressure housing. If images were processed and stored in the same housing as the camera, this scenario would limit the required information transfer between pressure housings on vehicles, which are often highly modular, only sharing compressed summary images and the accompanying semantic map. This implementation also makes the camera unit more modular and applicable to other monitoring applications such as moored or cabled observatories that are continuously collecting image data and storage constraints become problematic over long timescales. Another way we could utilize acoustic modems and image compression techniques to reduce the latency of understanding is to continuously transmit a low-bandwidth descriptor for each image as it is captured. This concept has roots in the mobile visual search paradigm where a descriptor is sent to a server in place

of the query image itself. In this scenario, the online clustering (or even repeated offline clusterings) would be performed on the ship where power and computational resources can be virtually unlimited and thus a better set of representative cluster centers can be obtained. The vehicle can then be queried to compress and transmit these representative images during the mission.

Kaeli [22] also compares JPEG, JPEG-2000 and SPIHT. JPEG is a common example of a lossy compression format which uses the DCT for each 8x8 block to achieve roughly 10:1 compression without major perceptual changes in the image. Recent work [22] has focused on using similar wavelet decomposition techniques for underwater applications using smaller packet sizes with the SPIHT [25] image compression algorithm, which generates a bit-oriented output stream and allows for its truncation at any point (exact output size). These methods are capable of acoustically transmitting one 1024x1024 color image in under 15 minutes. In almost all circumstances, the aspects of an image that are important to the user can be conveyed using many less bytes than are in the original image. This compressed image can then be transmitted more efficiently across a network using less bandwidth. In lossless compression, the original image can be fully recovered from the compressed format, where in lossy compression it cannot. Lossy compression techniques are capable of higher compression rates than lossless techniques at the expense of making assumptions about what the user deems important. It is designed largely on models of human visual perception, so some of the artifacts make it ill-suited for image processing applications. Every 3 minutes the most recent image was compressed and queued for transmission, ensuring there would always be available imagery to transmit and providing the operator with a naive understanding of the survey environment. Related work in online data summaries focuses on determining a small subset of images that best represent a collection of images. At sub-image scales, saliency-based methods can recommend regions of interest within an image for preferential transmission.

2.2 Image Compression

The traditional approach to image compression is composed of the following steps [32]:

1. Transformation: the original coefficients undergo a reversible transformation producing new coefficients that are uncorrelated and independently distributed. It is usually assumed that there is no information loss in this step but many transforms may actually introduce rounding errors and may produce small information losses.

2. Quantization: The transformed coefficients are quantized producing a large number of null (zero) coefficients for coarse quantization or a large number of small coefficients for finer quantization. In the limit of no quantization (lossless), the transformed coefficients are expected to have a lower entropy than the original ones in any case, so compression should still be possible but with a smaller ratio.

3. Entropy coding: Entropy coding techniques are applied to the quantized coefficients, assigning a small entropy value (high probability) to a null coefficient. This step, together with the transformation step, is usually the most time consuming, specially when complex statistical modelling of the source coefficients are used to assign dynamic probabilities which, in turn, guide the operation of an entropy coder, e.g., arithmetic coder.

The widely used JPEG [24] image compression algorithm is a classic example of this class of coders but performs extremely badly in low bit rate scenarios (it was never designed for this use case). Also, JPEG [24] is a class of algorithms that compresses up to a certain "quality" value, i.e., the user cannot specify a specific target rate and obtain a stream with a precise predefined size and does not have a lossless mode (JPEG-LS [39] is an entirely different algorithm that only shares the name with it and is not a transform coder).

The steps performed by the classic JPEG [24] algorithm usually follow a predetermined spatial scanning order of the coefficients and the resulting compressed stream cannot simply be truncated in order to get a lower quality version of the original image.

Given a distortion metric, low bit rate compression requires that coefficients be scanned in decreasing order of distortion reduction and not in a simple predefined spatial order like in JPEG [24]. This scanning order is dependent on the contents of the image being compressed and how to convey the chosen order (addressing information or significance map) is a major difficulty in the design of an efficient coder.

Multi-scale image representation and progressive transmission are not new, dating back to laplacian pyramids [40] which are overcomplete representations and do not constitute a proper basis [41]. However, very efficient embedded coding which could rival non-embedded ones was achieved by the introduction of the Embedded Zerotree Wavelet (EZW) [26] algorithm, which coupled the critically sampled Discrete Wavelet Transform (DWT) [42] with bitplane coding. EZW [26] also introduced the concept of zerotrees allowing for the efficient coding of the addressing information of the next coefficient to be encoded (significance map).

Most state-of-the-art progressive bit oriented embedded algorithms, like the Set Partitioning In Hierarchical Trees (SPIHT) [25] and the Set Partitioning Embedded bloCK coder (SPECK) [16] are based and improve on the original EZW [26] and introduce a very significant difference - the entropy coding step can be omitted with a only small penalty in the final compression ratio, allowing for very fast implementations.

The new standard JPEG-2000 [23] algorithm is a state-of-the-art image compression algorithm and addresses most shortcomings of the original JPEG [24] algorithm including a progressive lossless mode. It uses the discrete wavelet transform and bitplane encoding coupled with Embedded Block Coding with Optimized Truncation (EBCOT) [43] and sophisticated Context Adaptive Binary Arithmetic Coding (CABAC) [23] to produce very high quality progressively compressed images. However, its output stream is composed of relatively large packets, i.e., it does not produce a bit oriented embedded stream, cannot be truncated at any point, does not perform well under extremely low bit rates, is complex, slow, highly dependent on the entropy coding step, and works with packet sizes which are not suited for underwater communications [22]. JPEG-2000 [23] also provides for Region Of Interest (ROI) [23] coding allowing for parts of the image to be encoded with higher priority than the rest, however, the standard only includes the “maxshift” (ROI map not required) and the “scaling” (ROI map required) ROI methods. The first (“maxshift”) is not really applicable in most situations, specially for low bitrate compression because it completely transmits the ROI region before any background information. The second (“scaling”) fares better but also insists on transmitting the lower ROI bitplanes blended with the higher background ones, wasting too many bits on usually noisy ROI information. Unless the decoder actually needs the lower bitplanes of the ROI region, it is usually better to delay their encoding for higher bit rates and better reproduce the background at lower bit rates.

Unlike JPEG-2000, however, none of the set partitioning algorithms claim rate distortion optimization for their output stream. Also, almost none of them have neither Region Of Interest (ROI) coding nor a lossless mode because they are heavily dependent on orthogonal transforms or the floating point biorthogonal Cohen-Daubechies-Feauveau CDF-9/7 DWT [44] with a $\langle\sqrt{2}, \sqrt{2}\rangle$ normalization which, in this case, presents almost unitary subband gains. The reason for this is that all current set-partitioning algorithms simply search for the greatest coefficient to code next, independently of its probability, entropy, position, and subband, assuming that this is the one that will reduce distortion the most. Almost all of them use sets of coefficients grouped into trees that span different subbands across the same spatial location and this data structure works best exactly when we are searching for significant coefficients among the coefficients that are part of the set, i.e., in any subband and, therefore, they work better when the all subbands' weights are one, which is not the case for reversible DWTs, i.e., DWT that maps integers into integers [45], or non expansive DWT.

In general, image compression algorithms are not specifically designed to handle extremely low bit rates and there is not a single solution which covers all the needs of underwater missions, e.g., SPIHT [25] does not possess neither lossless mode nor ROI coding and even though it compresses very well it is not very efficient for extremely low bit rates due to its initial high number of sets (which is addressed in SPECK [16] and WBTC [28] algorithms to a certain extent).

2.3 Parallelization Schemes for the DWT

The Discrete Wavelet Transform (DWT) implementation has been extensively studied and many algorithms for its efficient implementation have been devised. With the advent of the lifting scheme [46–48], an efficient and inplace implementation became possible and it has also been the subject of extensive research with many published results on its implementation on various platforms, ranging from application specific integrated circuits (ASIC), field programmable gate arrays (FPGA), general purpose graphics processing units (GPGPU) and single and multi core general purpose processors (CPU) [49–61].

Hutcheson and Natoli [62] shows that the memory bandwidth increase due to caching is greater when memory is accessed sequentially, that the parallelization of an algorithm can yield a significant increase in effective bandwidth, and that random memory access eliminates most of the benefits of caching, resulting in a decrease of the maximum bandwidth available. As the computational intensity of a program increases, it becomes less limited by memory bandwidth and more limited by the processors arithmetic performance. A modified version of the STREAM [63] benchmark was used. Using OpenMP, programs can be parallelized, so that more than one core or processor are utilized at once. The bandwidth benefit from L1 and L2 caching scales in a linear fashion as the number of cores is increased, and the bandwidth benefit from L3 scales linearly as the number of processors is increased. Therefore, parallelization of an algorithm can yield a significant increase in effective bandwidth. The greater the computational intensity, the less benefit is experienced from caching, until a main memory call eventually takes less time than the processing of the data. A working knowledge of the caching process, as well as the factors that limit the speed of an algorithm, including bandwidth and computational intensity, can allow a programmer to better optimize their code to maximize use of limited system resources. With this knowledge, strategies such as parallelization can be used effectively to improve algorithms and achieve faster runtimes.

Verbanescu [48] proposes a framework for the efficient programming of parallel applications on multi-core processors, summarize these challenges, and verify if any of the modern programming tools, mostly designed to support multi-cores, offer the right mix of features to support the user in the whole development process, and classifies the programming models into two disjoint classes: application-centric and hardware-centric.

Chaver et al. [56, 58] presents an implementation of a 2-D Discrete Wavelet Transform on general-purpose microprocessors, focusing on both memory hierarchy and SIMD parallelization issues. Both topics are somewhat related, since SIMD extensions are only useful if the memory hierarchy is efficiently exploited. In this work, locality has been significantly improved by means of a novel approach called pipelined computation, which complements previous techniques based on loop tiling and non-linear layouts.

Barina et al. [47, 57] presents an efficient computation of the two-dimensional discrete wavelet transform. The state-of-the-art methods are extended in several ways to perform

the transform in a single loop, possibly in multi-scale fashion, using a compact streaming core. The approach presented fits nicely into common SIMD extensions, exploits the cache hierarchy of modern general-purpose processors, and is suitable for parallel evaluation. This core can further be appropriately reorganized to target the minimization of certain platform resources. Finally, the approach presented is incorporated into the JPEG 2000 compression chain, in which it has proved to be fundamentally faster than widely used implementations (the thesis focuses on the CDF (Cohen-Daubechies-Feauveau) 5/3 and 9/7 wavelets only).

Barina et al. [64] describes the DWT computation as well as Mallat's filtering scheme, and shows that the lifting scheme [46] is usually faster. The lifting data flow graph consists of a regular grid computational scheme suitable for SIMD vectorization. Both algorithms can be performed over some approximation of real numbers focuses on single-precision floating-point format. In contemporary personal computers (PCs), the general purpose microprocessor with SIMD instruction set is often found. In case of the x86-64 architecture, the appropriate instruction set used here is SSE (Streaming SIMD Extensions).

Bo-Cheng et al. [49] introduces a hybrid multi threaded object detection with high parallelism and extensive data reuse to ARM processors. A self adaptable flow is proposed to adjust the multi threaded object detection to fully exploit various embedded multi core architectures. The ARM-based cycle accurate simulations of multicore systems have shown the superior performance returned by the proposed design.

Cheng et. al. [50] presents an efficient array architectures for multi-dimensional (m-D) discrete wavelet transform (DWT) VLSI implementation, in which the lifting scheme of DWT is used to efficiently reduce hardware complexity. The parallelism of 2m subbands transforms in lifting-based m-D DWT is explored, which efficiently increases the throughput rate of separable m-D DWT with fewer additional hardware overhead. The proposed architecture is composed of m2m1 1-D DWT modules working in parallel and pipelined, which is designed to process 2m input samples per clock cycle, and generate 2m subbands coefficients synchronously.

Colom-Palero et al. [51] presents a flexible filter and control unit structure for implementing different VLSI architectures on two-dimensional DWT. These structures are applied over three different architectures: a direct approach, the Recursive Pyramidal

Algorithm (RPA) architecture, and a new proposed modification of RPA. This modified architecture works in a non-separable fashion using a parallel filter structure with distributed control to compute all the DWT resolution levels. It is fully modular and scalable, with low latency and high throughput performance. Implementation results based on a Virtex-II FPGA device are included. Real-time video processing is achieved.

Chapter 3

Minimal Time DWT Algorithm

Most transform based compression algorithms spend a substantial amount of time in the computation of the transform itself [65]. Multiresolution algorithms are at a disadvantage in relation to block transform ones due to their need to go over the whole data many times and with different memory access patterns.

Current state-of-the-art compression algorithms, specially embedded ones, use the critically sampled Discrete Wavelet Transform (DWT) in order to get an energy-compacted and multi-resolution representation of the original signal so that the entropy of the transformed signal is smaller than the original one, allowing for compression of the original signal in the transform domain.

The current literature regarding DWT implementations is, however, somewhat contradictory in the sense that some sources cite the DWT lifting implementation as being memory bound [54] while others state that some parallel algorithms scale linearly with the number of cores [66], which are in conflict unless the parallel implementations are far from saturating the memory bus (which seems to be the case).

In fact, the 2-D DWT inplace lifting transform, composed of alternating predict and update steps on lines and columns, is quite simple to vectorize, allowing for an efficient implementation with little effort. This led to the study of memory bound algorithms on multicore processors presented here and how to efficiently explore the memory cache hierarchy in a multithreaded environment in order to achieve an optimum (minimal time) 2-D DWT algorithm.

It must be observed that memory bound algorithms, however, differ in the number of main memory operations (read and write) and their respective memory access patterns, e.g., an algorithm that performs 2 main memory operations has the potential of being twice as fast as another that performs 4 main memory operations, for example.

The idea is to explore the memory cache hierarchy, which is present in all current general purpose processors, in order to minimize the number of main memory operations, i.e., the amount of times an algorithm needs to actually read or write to a main memory location and consequently maximize its cache utilization.

A general scheme for the computation of the 2-D DWT is given which requires the minimum number of main memory accesses, resulting in minimal running time depending on the number of cores, their relative performance in relation to the memory subsystem, and the quality of the implementation.

3.1 Introduction

This chapter focuses on general purpose processors [54, 57] which may contain one or more cores and a certain amount of fast data cache memory shared among all cores, usually referred to as the level 2 (L2) cache [62, 63, 67]. Almost all modern CPUs contain a fast level 1 (L1) cache exclusive to each core (non-shared) which is usually split between instruction and data caches. Some more advanced CPUs also offer a larger but slower data L2 cache exclusive to each core (non-shared) and then an even larger and slower level 3 (L3) data cache shared among all cores. For the purposes of this paper, cache will refer to the shared data cache, irrespective if it is actually the second or third level cache of the current architecture. We also use the term “memory operation” (MOP) to refer to either a read from or write to an actual main memory location.

For most DWT implemented via lifting (Figure 3.1), each predict and update step is usually composed of 2 additions and 1 multiplication or, if a fused multiply-add (FMA) or fused multiply-subtract (FMS) instruction is available, 1 addition and 1 FMA or 1 FMS instruction [32]. Therefore, a “good” implementation of the predict and update steps will most likely be limited by the available memory bandwidth [62, 67] and their parallel implementation will most likely not scale with the number of threads [46, 53, 55, 57].

It is well known that the lifting DWT algorithm is memory bound [60], consisting of simple and easily vectorizable predict and update steps.

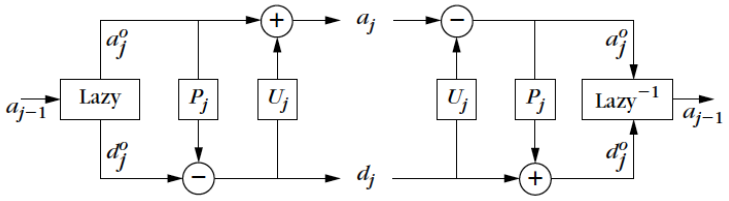


FIGURE 3.1: Lifting: Predict and Update (forward and reverse)

Memory bound algorithms, however, differ in how many MOPs they actually perform, e.g., a memory bound algorithm with only 2 MOPs per coefficient has the potential of being twice as fast as another with 4 MOPs, irrespective of the number of cores. Memory bound algorithms, once having reached the point of memory saturation with a certain number of cores, will not perceive any increase and may even have its performance decreased by using more cores due to the scattered memory access patterns produced by the parallel execution of its threads.

Most existing multithreaded lifting algorithms, if not all, rely on the disjoint partitioning of the image data so that each thread works on its own independent partition and does not have to do any costly synchronization [68] among them except their creation and joining at the beginning and end of each parallel block [47, 52, 64], respectively. However, such simple data partitioning fails to efficiently explore the memory hierarchy and its cache levels, which are crucial for a fast implementation and end up performing more MOPs than the minimum necessary due to cache misses.

Due to the dependency of the lifting steps (Figure 3.1), i.e., the update steps use the results of the previous predict steps, and the dependency of the line and column transformations, i.e., lines before columns or vice-versa, some form of efficient coordination among the threads is necessary in order to keep them clustered and, at the same time, use the maximum amount of CPU cycles on actual work which contributes to the solution instead of synchronization among them.

The algorithm presented here explores both spatial and temporal data locality in a multithreaded environment by using an efficient thread synchronization scheme and achieves the minimum number of MOPs possible (one memory read and one memory write per coefficient), i.e., reaches the minimum latency possible if its implementation is

capable of saturating the memory bus, or a path to this minimum by using more cores if the implementation or the cores are not fast enough.

Most claims of linear performance gains with the number of threads seem to rely on inefficient implementations and may be misleading. In fact, it is much easier to saturate the memory bus if the algorithm does more than the minimum number of MOPs and, therefore, memory saturation by itself is not a good indication of the quality of an algorithm but only of the quality of its implementation.

3.2 Cache Memory Architecture Overview

A cache memory is a block of memory available to the processor in order to reduce the time to access data from the main memory. The cache is a faster and smaller memory which stores copies of the data from frequently used main memory locations. Most processors contain different independent caches [62], usually composed of a level 1 (L1) cache split between instructions and data for each core and a hierarchy of progressively slower and larger data caches of which at least the last level(s) is(are) shared among all cores.

Cache performance has become extremely important due to the exponential increase in the speed gap (Figure 3.2) between the processor and main memory [69]. In this figure the aggregate CPU performance (Standard Performance Evaluation Corporation - SPEC) and the memory access speed (inverse latency) are normalized to 1 in 1980 and their respective growth is compared, showing that their performance difference has been exponentially increasing. In fact, processor development has been focused on speed while memory development has been focused on capacity, which partly explains the different growth rates.

While processors can access data at byte granularity, the physical access to memory is usually done in aligned fixed size blocks called cache lines [62, 67]. While most embedded processors have a single 32-bit memory channel interface, most desktop processors access memory via 2 or more 64-bit channels allowing for greater bandwidth and consequently, greater performance, so that the much faster processor does not starve for data for most common workloads. As a side note, most current Intel and ARM processors usually use a data cache line which is 64 bytes long.

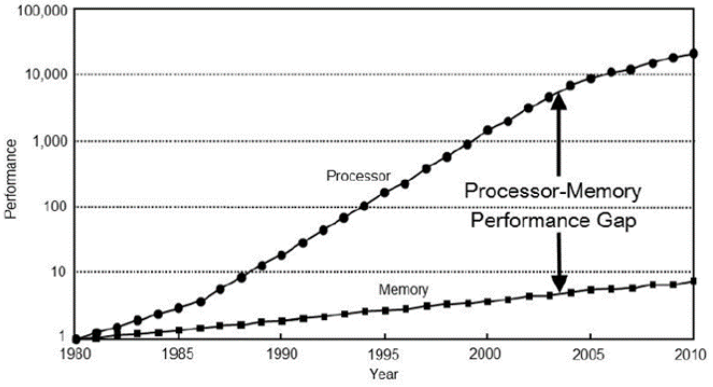


FIGURE 3.2: Normalized Processor(SPEC)×Memory(Latency^{-1}) performance gap

In all cases, memory access patterns also play a major role in the total memory bandwidth available to applications, benefiting those applications that read data sequentially (burst reads) and make good use of the hardware prefetcher, as opposed to those applications that read memory in either random or strided patterns [62].

Whenever a core needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache (assuming a write-back cache policy). If so, there is a cache hit and the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory. When the memory location is not in the cache there is a cache miss and the processor must first allocate a cache line entry (and possibly evict some dirty preexisting cache line by writing it to main memory), read the contents of the corresponding cache line from main memory and only then proceed with the original operation from or to the cache.

Good knowledge of the inner workings of the cache memory subsystem is essential for the design of high performance software, specially for memory bound problems on both single and multi core processors [56, 58]. Exploration of both spatial and temporal locality is crucial in achieving performance gains and translate directly into cache hit and miss ratios.

The speed gap between processor and memory (Figure 3.2) has become so large that stalls are practically unavoidable, i.e., the core sits idle during the time taken to fetch one cache line from memory (read latency due to a cache miss). In order to keep the cores busy, most modern high performance processors allow out-of-order execution, in which each core attempts to execute future independent instructions while waiting for

the cache miss data, and simultaneous multithreading (SMT), which allows another alternate thread to use the core while the original thread waits for the required cache miss data, among other things [69, 70].

3.3 Memory Cost Model

Even though the concept of bandwidth from main memory given in bytes per second is quite an easy concept to grasp, the quantity lends itself to some different interpretations, depending on the level of abstraction used [62].

In order to establish our memory cost model, a metric that counts the actual number of bytes read or written by the hardware (hardware metric) is used for measuring the MOPs for the DWT algorithms. Cache accesses are ignored, assuming that their cost is much smaller than the cost of accessing the main memory.

Also, a write back cache policy is assumed so that a write to a memory location may have to fetch its corresponding cache line and then write the new value to the cache, which will eventually have to be evicted and finally written to memory at some later time. It should be noted that, while the cache line is active, multiple reads and writes to the same memory locations will have no cost (according to our model) and only the cost of the final eviction will count for those locations that were written. For example, assuming no memory locations are currently cached, the assignment $a = b$ will have a cost of 3 MOPs which are the 2 cache line fetches corresponding to the locations a and b and 1 cache line write when the cache line corresponding to a gets evicted from the cache at a later time. All operations done on a while it is still cached will essentially have no cost, e.g., the assignment $a = a + b$ will have no cost.

With this in mind, the STREAM [63] benchmark was modified to report the hardware memory bandwidth, assigning a value of 2 instead of 1 to an uncached memory write for all kernels. Also, two other kernels called “inlift” and “lift”, composed of the array operations $a[j] = a[j] - scalar * (b[j] + c[j])$ and $d[j] = a[j] - scalar * (b[j] + c[j])$, respectively, were added to the existing ones in order to verify the hardware bandwidth of an inplace lifting step for the fist and of a lifting step to a different location for the latter.

3.4 DWT Lifting Algorithms

There are many ways to actually implement the separable 2-D DWT lifting transform inplace, extensively covered in the literature, essentially differing on how and when to do the column transformations [56, 57].

It should be noted that the order of the 1-D transformations on lines and columns matters, specially for the integer transforms which use integer arithmetic and truncation, which is a non linear operation. In other words, if the forward transform is chosen to first transform the lines and then the columns, the reverse transform should first transform the columns and then the lines, and vice-versa.

In order to simplify the presentation, the algorithms presented here assume an integer 5/3 biorthogonal interpolating transform (ibior5x3) which is described by its predict step

$$d_j^1 = d_j^0 - \left\lfloor \frac{s_j^0 + s_{j+1}^0}{2} + \frac{1}{2} \right\rfloor \tag{3.1}$$

followed by its update step

$$s_j^1 = s_j^0 + \left\lfloor \frac{d_{j-1}^1 + d_j^1}{4} + \frac{1}{2} \right\rfloor \tag{3.2}$$

where $\lfloor x \rfloor$ stands for the largest integer smaller than or equal to x (floor).

In these lifting equations, the superscript notation indicates the iteration number and it is assumed that the original signal $\{x_j\}$ has been split into even $s_j^0 = x_{2j}$ and odd $d_j^0 = x_{2j+1}$ samples (lazy wavelet transform).

It is also assumed that the forward transform operates on lines and then on columns, and that the images are interleaved and larger than twice the shared cache size. With these assumptions, our memory cost model applies and the number of MOPs for each algorithm can be calculated. Once the number of MOPs of an algorithm is found, it can be used to compare its performance relative to other algorithms and its minimum execution time can be established based on the maximum memory bandwidth available to the processor.

Algorithm 1 defines some types and macros that are used throughout. If the exactly same transform used by the JPEG2000 [23] algorithm is desired, the predict step in (3.1)

should remove the addition of the term 1/2 from the right hand side, i.e., the predict step should be substituted by the simpler "`#define P(a,b) (((a) + (b)) >> 1)`".

Algorithm 1 Macros and Definitions

```
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_num_threads() 1
#endif

typedef short coeff_t;
typedef char atom;
typedef volatile atom atomic; // _Atomic
#define ATOMIC_VAR_INIT(val) (val)
#define atomic_load_acq(ptr) (*(ptr)) // acquire
#define atomic_store_rel(ptr,val) (*(ptr)=(val)) // release

#define CAS(ptr,old,new) __sync_val_compare_and_swap(ptr,old,new)

#define NONE    0
#define BUSY    1
#define DONE    2
#define WORK    4
#define MORE    8

typedef struct shrState {
  int     height;
  int     ntypes;
  atomic  array[];
} shrState_t;

inline static atomic *
shrstate(shrState_t *ss,int type,int n)
{
  return &ss->array[((n * ss->ntypes) + type)];
}

#define _LINE   0
#define _COLUMN 1
#define lnstate(s,n) shrstate(s,_LINE,n)
#define clstate(s,n) shrstate(s,_COLUMN,n)

#define P(a,b)  ((((a) + (b)) + 1) >> 1) // predict
#define U(a,b)  ((((a) + (b)) + 2) >> 2) // update

// interleaved even sample
#define S(n)    s[((n) << 1)]
#define _S(n)   s[((n) << 1) * w]
// interleaved odd sample
#define D(n)    s[((n) << 1) + 1]
#define _D(n)   s[(((n) << 1) + 1) * w]
```

In all algorithms presented here, the line transformation which is described in Algorithm 2 does exactly 1 read and 1 write per coefficient (the code presented takes into consideration the operation on the borders). In fact, the vector implementation loads each vector of coefficients from memory only once, does the necessary deinterleaving, computation, and storage of the necessary intermediate results using only the vector registers inside the processor's cores and finally only writes the resultant transformed vector. It should be noted that the write done is to a value that is already in the cache so only the final

write counts as a MOP. Therefore, the total number of MOPs for the line transform is 2, which is the smallest number possible.

Algorithm 2 Line predict-update

```
static void
fwd_ln_inp(coeff_t *s,int width)
{
  if (width > 2) {
    D(0) -= P(S(0),S(1));
    S(0) += U(D(0),D(0));
    coeff_t *z;
    for (z = s + (width - 4); s < z; s += 2) {
      D(1) -= P(S(1),S(2));
      S(1) += U(D(0),D(1));
    }
    if (s == z) {
      D(1) -= P(S(1),S(1));
      S(1) += U(D(0),D(1));
    } else {
      S(1) += U(D(0),D(0));
    }
  } else if (width > 1) {
    D(0) -= P(S(0),S(0));
    S(0) += U(D(0),D(0));
  }
}
```

Also, for the column algorithms that will be presented, the array of pointers to the lines `v[]` has two more elements `v[-1]` and `v[height]` which have the same values as `v[1]` and `v[height-2]`, respectively, which takes care of the borders and simplify the algorithms.

3.4.1 Vertical Naive Algorithm - cnaive

The definition of the separable 2-D DWT directly implies a straightforward implementation, i.e., implement the lifting steps on all lines followed by the lifting steps on all columns.

In this case, each column transformation, shown in Algorithm 3, uses a strided access pattern but, just like the line transformation, only does 1 read and 1 write to a location which is already in the cache for a total of 2 MOPs.

In total, the cnaive algorithm needs 4 MOPs (2 for the lines and 2 for the columns) in the best case. However, due to the strided nature of its memory access, it is hard for this algorithm to achieve the theoretical maximum memory bandwidth available to the processor because the elements in a column may all reside in a single channel of the memory subsystem and also because it does not take advantage of neither prefetching nor

Algorithm 3 Vertical Column predict-update inplace

```
static void
fwd_cl_inp(coeff_t *s,int height,int stride)
{
  int w = stride;
  if (height > 2) {
    _D(0) -= P(_S(0),_S(1));
    _S(0) += U(_D(0),_D(0));
    coeff_t *z;
    for (z = s + (height-4) * w; s < z; s += 2 * w) {
      _D(1) -= P(_S(1),_S(2));
      _S(1) += U(_D(0),_D(1));
    }
    if (s == z) {
      _D(1) -= P(_S(1),_S(1));
      _S(1) += U(_D(0),_D(1));
    } else {
      _S(1) += U(_D(0),_D(0));
    }
  } else if (height > 1) {
    _D(0) -= P(_S(0),_S(0));
    _S(0) += U(_D(0),_D(0));
  }
}
```

memory read bursts, which are crucial to achieve the theoretical maximum bandwidth available.

Nonetheless, a multi threaded version of this algorithm may get close to the maximum available bandwidth in certain cases but in general this is not easily predictable.

The final cnaive algorithm is shown in Algorithm 4 and is composed of 2 simple loops, the first for transforming the lines and the second for the columns [56, 57].

Algorithm 4 Vertical naive algorithm - cnaive

```
int
fwd_cnaive_mx_inp(coeff_t *v[],int width,int height,int stride)
{
  for (int n = 0; n < height; ++n)
    fwd_ln_inp(v[n],width);
  for (int n = 0; n < width; ++n)
    fwd_cl_inp(v[0]+n,height,stride);
  return height;
}
```

3.4.2 Horizontal Naive Algorithm - lnaive

The strided access pattern used by the column transformation can be eliminated and turned into a sequential pattern by separating the predict and update steps for the columns, i.e., first do all the predict steps for all columns for each odd line (Algorithm 5), followed by all the update steps for all columns for each even line (Algorithm 6).

Algorithm 5 Horizontal Column predict

```
static void
fwd_d_cl_inp(coeff_t *v[],int width)
{
  coeff_t *prev = v[-1];
  coeff_t *curr = v[ 0];
  coeff_t *next = v[ 1];
  for (int n = 0; n < width; ++n)
    curr[n] -= P(prev[n],next[n]);
}
```

Algorithm 6 Horizontal Column update

```
static void
fwd_s_cl_inp(coeff_t *v[],int width)
{
  coeff_t *prev = v[-1];
  coeff_t *curr = v[ 0];
  coeff_t *next = v[ 1];
  for (int n = 0; n < width; ++n)
    curr[n] += U(prev[n],next[n]);
}
```

For each predict and update step there will be 2 new reads (the current and next lines are new but the last line is in the cache) and 1 final write (current line) which amounts for 3 MOPs for each step. As each step is applied to half of the lines (predict on odd and update on even), there is a total of $3/2 + 3/2 = 3$ MOPs for the whole column transformation.

In total, the lnaive algorithm needs 5 MOPs (2 for the lines and 3 for the columns) in the best case and has a sequential memory access pattern.

Algorithm 7 shows its implementation which is composed of 3 loops, the first for the lines, the second for the horizontal predict of the odd lines, and the third for the horizontal update of the even lines [56, 57].

Algorithm 7 Horizontal naive algorithm - lnaive

```
int
fwd_lnaive_mx_inp(coeff_t *v[],int width,int height)
{
  for (int n = 0; n < height; ++n)
    fwd_ln_inp(v[n],width);
  if (height > 1) {
    for (int n = 1; n < height; n += 2)
      fwd_d_cl_inp(v + n,width);
    for (int n = 0; n < height; n += 2)
      fwd_s_cl_inp(v + n,width);
  }
  return height;
}
```

3.4.3 Pipelined Algorithm - pipeline

The pipeline algorithm is a well known algorithm [53, 56] and is a direct consequence of a memory optimization applied to the lnaive algorithm [57]. From a top-down perspective of the lnaive algorithm, each horizontal column update needs to have all horizontal column predicts done up to its next line which, in turn, needs to have all line transformations done up to its next line. Therefore, the pipeline algorithm (Algorithm 8) is implemented by interleaving 2 line transformations (current and next), 1 horizontal column predict (current) and 1 horizontal column update (previous).

Algorithm 8 Pipelined algorithm - pipeline

```
int
fwd_pipeline_mx_inp(coeff_t *v[],int width,int height)
{
  if (height > 0) {
    fwd_ln_inp(v[0],width);
    if (height > 1) {
      int last = height - 1;
      for (int n = 1; n < last; n += 2) {
        fwd_ln_inp(v[n],width);
        fwd_ln_inp(v[n + 1],width);
        fwd_d_cl_inp(v + n,width);
        fwd_s_cl_inp(v + (n - 1),width);
      }
      if (last & 1) {
        fwd_ln_inp(v[last],width);
        fwd_d_cl_inp(v + last,width);
        fwd_s_cl_inp(v + (last - 1),width);
      } else {
        fwd_s_cl_inp(v + last,width);
      }
    }
  }
  return height;
}
```

As always, each line transformation accounts for 2 MOPs and, assuming the last 4 lines are in the cache, the current horizontal column predict and previous horizontal column update are free because all values read and written are already cached and none of them have been evicted yet. In total, **the pipeline algorithm needs only 2 MOPs for both lines and columns, i.e., for the whole transform, reaching the theoretical minimum number of MOPs which is one read and one write per element**.

3.5 DWT Parallel Lifting Algorithms

A brief description of the parallelization of the naive algorithms is given followed by a detailed description of the parallel pipelined (paraline) algorithm.

3.5.1 Parallel Naive Algorithms

Parallel versions of both the cnaive and lnaive algorithms have been the subject of many studies in the literature [47, 56, 57] and are quite simple to derive from their single threaded versions by simply substituting each `for` loop by its corresponding parallel version. For example, using the Open Multi Processing `OpenMP` Application Programming Interface (API), the insertion of a line containing "`#pragma omp parallel for`" immediately before each `for` loop in algorithms 4 and 7 is all that is necessary to obtain a working parallel version for each algorithm. For the `CilkPlus` C language extension, all that is needed is to substitute the keyword `for` in Algorithms 4 and 7 by the keyword `cilk_for`.

3.5.2 The Parallel Pipelined Algorithm - paraline

According to our simplified memory model, the pipeline algorithm uses 2 MOPs irrespective of the complexity of the DWT and, while the cnaive algorithm also uses a fixed number of 4 MOPs, also irrespective of the DWT complexity, its memory access pattern is not sequential which usually makes it the worst performing algorithm, even when comparing it with the lnaive algorithm which uses more MOPs - 5 for the ibior5x3 but more for more complex DWTs which have longer filter lengths and/or more lifting steps.

As the pipeline algorithm uses only 2 MOPs, it is theoretically possible to achieve a performance increase of $5/2 = 2.5$ in relation to the lnaive algorithm for the ibior5x3.

In fact, as 2 is the minimum number of MOPs possible, if the parallel pipelined algorithm implementation is capable of saturating the memory bus with the available number of cores then we have reached the maximum performance achievable on this system, i.e., the minimum execution time. From here, the only option to increase performance, in this case, is to use a faster memory subsystem.

The main contribution of this chapter is the parallel version of the pipeline algorithm, called the paraline algorithm, which is not as straightforward to derive due to the dependencies on the order of execution of the line, column predict, and column update steps. The challenge is to keep all threads busy working on the solution of the problem (never neither busy waiting nor sleeping) without wandering too far away from each other so that the original assumption of only 2 MOPs holds for the whole transform.

The paraline algorithm was designed to be lock free, i.e., to not make use of any locks, so it is guaranteed to never sleep waiting on a condition while scaling well with the number of threads. It does, however, makes use of the Compare And Swap (CAS) atomic primitive, available on most processors in order to synchronize the multiple execution threads.

The CAS operation compares the contents of a memory location to a given value and, in case they are equal, modifies the contents of that memory location to a given new value. This is done as a single atomic operation (Algorithm 9) and in the actual code we make use of a builtin function of the GNU Compiler Collection (GCC) C compiler which does exactly that, as described in Algorithm 1 for the macro `CAS`.

Algorithm 9 CAS primitive pseudo-code

```
Atomic CAS(valptr,oldval,newval)
{
    val = *valptr;
    if (val == oldval)
        *valptr = newval;
    return val;
}
```

Also, at least acquire-release semantics [71] is assumed for the underlying hardware memory model so that both `atomic_load_acq` and `atomic_store_rel` (Algorithm 1) actually synchronize the memory dependencies among the threads. As a note, this is true for current x86_64 processors from Intel but not for the ARM architecture, which requires memory barriers in order to guarantee that a thread sees the loads and stores of other threads in the correct order which, along with the `CAS` operation, are implemented in Algorithm 16 using the standard header "`stdatomic.h`".

As always, there is some overhead when using threads which, at the very least, accounts for the thread creation and joining for each parallel section (2 for the cnaive and 3 for the lnaive algorithms). Even though the paraline algorithm has only 1 thread creation and joining section, it uses some state, shared among all threads, regarding the status of each line, column predict, and column update. Each thread has to check the status multiple times and eventually perform some CAS operations in order to guarantee the correct execution order.

The best way to describe the paraline algorithm is probably using the top-down approach. For the ibior5x3, in order for a horizontal column update to be completed

(Algorithm 12), both its previous and next lines must have their horizontal column predict completed and, for each horizontal column predict (Algorithm 11), its previous, current, and next lines must have their line transformation completed (Algorithm 10).

Algorithm 10 Threaded line predict-update

```
static atom
_try_fwd_ln_inp(coeff_t *src,int width,atomic *sp)
{
  atom s = CAS(sp,NONE,BUSY);
  if (s != NONE)
    return s;
  fwd_ln_inp(src,width);
  atomic_store_rel(sp,DONE);
  return (DONE | WORK);
}

inline static atom
try_fwd_ln_inp(coeff_t *src,int width,atomic *sp)
{
  atom s = atomic_load_acq(sp);
  return (s != NONE) ? s : _try_fwd_ln_inp(src,width,sp);
}
```

Algorithm 11 Threaded column predict (n is odd)

```
static atom
_try_fwd_d_cl_inp(coeff_t *v[],int width,int height,shrState_t *state,int n)
{
  atom s = try_fwd_ln_inp(v[n-1],width,lnstate(state,n-1));
  if (n < (height - 1))
    s |= try_fwd_ln_inp(v[n+1],width,lnstate(state,n+1));
  if (s & BUSY)
    return (BUSY | (s & WORK) | MORE);
  atomic *sp = clstate(state,n);
  atom z = CAS(sp,NONE,BUSY);
  if (z != NONE)
    return (z | (s & WORK));
  fwd_ln_inp(v[n],width);
  fwd_d_cl_inp(v + n,width);
  atomic_store_rel(sp,DONE);
  return (DONE | WORK);
}

inline static atom
try_fwd_d_cl_inp(coeff_t *v[],int width,int height,shrState_t *state,int n)
{
  atom s = atomic_load_acq(clstate(state,n));
  return (s != NONE) ? s : _try_fwd_d_cl_inp(v,width,height,state,n);
}
```

This leads to a simple algorithm in which at the highest level there is a single loop that calls the horizontal column update step on the even lines (Algorithm 13).

Algorithms 10, 11, and 12 are presented with an associated inline function that should reduce the number of CAS operations performed but it is an optimization and is not needed in order to present the algorithm. In fact, it could have either been incorporated into each function itself or even be eliminated altogether.

Algorithm 12 Threaded column update (n is even)

```
static atom
_try_fwd_s_cl_inp(coeff_t *v[],int width,int height,shrState_t *state,int n)
{
  atom s = NONE;
  if (n > 0)
    s |= try_fwd_d_cl_inp(v,width,height,state,n-1);
  if (n < (height-1))
    s |= try_fwd_d_cl_inp(v,width,height,state,n+1);
  if (s & BUSY)
    return (BUSY | (s & WORK) | MORE);
  atomic *sp = clstate(state,n);
  atom z = CAS(sp,NONE,BUSY);
  if (z != NONE)
    return (z | (s & WORK));
  fwd_s_cl_inp(v+n,width);
  atomic_store_rel(sp,DONE);
  return (DONE | WORK);
}

inline static atom
try_fwd_s_cl_inp(coeff_t *v[],int width,int height,shrState_t *state,int n)
{
  atom s = atomic_load_acq(clstate(state,n));
  return (s != NONE) ? s : _try_fwd_s_cl_inp(v,width,height,state,n);
}
```

Algorithm 13 Threaded pipeline algorithm

```
static int
try_fwd_mx_inp(coeff_t *v[],int width,int height,shrState_t *state,int first)
{
  int n = first;
  while (n < height) {
    atom s = try_fwd_s_cl_inp(v,width,height,state,n);
    if ((s & DONE) || !(s & MORE)) {
      n += 2;
    } else if (!(s & WORK)) {
      for (int m = n+2; (m < height) && !(s & WORK); m += 2)
        s |= try_fwd_s_cl_inp(v,width,height,state,m);
      if (!(s & WORK))
        return (s & MORE) ? n : height;
    }
  }
  return height;
}
```

In order to keep the threads working on adjacent lines as much as possible so that our in-cache assumption holds, each thread does some work and then tries to find its next job in the earliest possible region which still needs work to be done. The amount of work each thread does until it searches for more work is of the utmost importance in order to keep the overhead low and is called a Work Unit (WU). In the example shown, the line transformation, the horizontal column predict, and the horizontal column update are prime candidates for being a WU and, in this case, there is a total number of $2 \times$ `height` WUs to be executed in order to complete the whole image transformation.

Associated with each WU there is a state which can be either `NONE` (`0`), `BUSY` (`1`), or

`DONE` (`2`), meaning that this WU has not been worked on, is currently being worked on, or is done, respectively. In order to avoid busy waiting, guarantee that all WUs are executed, and try to keep all threads close to each other, two more flags are returned (but never stored) for any function call which tries to do any work, which is logically or'd to the state of the requested WU when returning from an execution attempt.

The first flag is the `WORK` (`4`) flag and its presence indicates to the caller that this thread has actually executed at least one WU, either directly or as the result of a dependency, i.e., it indicates whether the thread has done actual work, either a line transformation or a column predict or update.

The second flag is the `MORE` (`8`) flag (which could also be called `AGAIN`) which indicates that the requested WU could not be processed because there are unfinished dependencies which are being processed by other threads (are `BUSY`). In this case, the processing of this WU should be attempted again later. In this case, the thread tries to work on the next possible WU and, after it has actually done any work, retries the pending WU.

For each thread, in case the current WU is not `DONE` and the `MORE` flag is set, busy waiting is avoided by checking the `WORK` flag. If it is set, then the thread tries to process the same WU again otherwise it tries to process the next available WU until it actually does any work and, only then, retries the pending WU. In case it couldn't find any work to do, the thread simply ends and returns either the line number of the pending WU or `height` in case there is no pending WU, which means the transformation is complete (or being completed by another thread).

The main thread, shown in Algorithm 14, allocates and initializes the necessary state and makes use of the widely available `OpenMP` parallel extensions to create $N-1$ threads that, together with the main thread, execute Algorithm 13 starting with the first column update WU (`first=0`). It should be noted that most compilers require the use of an appropriate flag in order to enable the `OpenMP` extensions (the popular `GCC` compiler requires the `-fopenmp` flag).

When the main thread is finished, it checks its own as well as the other threads' return values. If at least one of them is greater than or equal to `height` then it is finished, otherwise, the main thread calls Algorithm 13 with the parameter `first` set to the maximum value returned from all previous threads (the latest WU line number from

Algorithm 14 Paraline algorithm

```
static shrState_t *
newSharedState(int height,int ntypes)
{
  int nwus = height * ntypes;
  shrState_t *ss = malloc(sizeof(shrState_t) + (nwus * sizeof(atomic)));
  if (ss) {
    ss->height = height;
    ss->ntypes = ntypes;
    for (int n = 0; n < nwus; ++n)
      ss->array[n] = ATOMIC_VAR_INIT(NONE);
  }
  return ss;
}

static void
delSharedState(shrState_t *ss)
{
  free(ss);
}

#define NSTATES 2 // ibior5x3

int
fwd_paraline_mx_inp(coeff_t *v[],int width,int height)
{
  int line = 0;
  shrState_t *state = newSharedState(height,NSTATES);
  if (state) {
#pragma omp parallel reduction(max:line)
    line = try_fwd_mx_inp(v,width,height,state,0);
    if (line < height)
      line = try_fwd_mx_inp(v,width,height,state,line);
  }
  delSharedState(state);
  return line;
}
```

which to start), which is strictly smaller than `height`. This will guarantee that the whole matrix has been transformed. A more rigorous analysis shows that this last step is not really necessary, i.e., one of the threads should always return a value which indicates that it has reached the end of the work, but the existing code was kept and is shown here.

Finally, Algorithm 15 shows the main function used to call either the pipeline or the paraline algorithms depending on the number of threads and a minimum problem size `OMP_MIN_SIZE` (left at zero here) where the fixed cost of starting and joining threads is not significant in relation to the time required to do the computation.

Algorithm 15 Main paraline algorithm

```
static coeff_t **
newMatrixBorder(coeff_t *buffer,int width,int height,int pitch,int border)
{
  coeff_t **lp = malloc((height + 2 * border) * sizeof(coeff_t *));
  if (lp) {
    coeff_t **v = lp + border;
    int last = height - 1;
    v[0] = buffer;
    for (int n = 1; n <= last; ++n)
      v[n] = (void *) v[n - 1] + pitch;
    for (int n = 1; n <= border; ++n) {
      v[-n] = v[n];
      v[last + n] = v[last - n];
    }
  }
  return lp;
}

static void
delMatrixBorder(coeff_t **lp)
{
  free(lp);
}

#define BORDER  1 // ibior5x3
#define OMP_MIN_SIZE 0 // define a sensible value for your platform

int
fwd_mx_inp(coeff_t *buffer,int width,int height,int pitch)
{
  if (height <= 0)
    return 0;
  if (height == 1) {
    fwd_ln_inp(buffer,width);
    return 1;
  }
  int line = 0;
  coeff_t **lp = newMatrixBorder(buffer,width,height,pitch,BORDER);
  coeff_t **v = lp + BORDER;
  if (lp) {
    int nthreads = 0;
#pragma omp parallel if(width * height > OMP_MIN_SIZE)
#pragma omp master
    nthreads = omp_get_num_threads();
    line = (nthreads == 1) ?
      fwd_pipeline_mx_inp(v,width,height) // single threaded
    :
      fwd_paraline_mx_inp(v,width,height); // multi threaded
  }
  delMatrixBorder(lp);
  return line;
}
```

3.6 Benchmark and Results

In order to test the paraline algorithm and assess the validity of our memory cost model, we have benchmarked our implementations on a 64-bit x86 Intel Desktop computer as well as on an low cost 32-bit ARM based Single Board Computer (SBC) Raspberry Pi 3 Model B (RPi3B) (this system is actually 64-bit capable but was used in 32-bit mode).

It should be noted that our implementation actually uses the single threaded version of each algorithm (with no synchronization overhead) whenever the number of threads is 1 and the values for the 1 thread case in all tables reflect this, e.g., the pipeline algorithm is used for the single threaded paraline algorithm.

3.6.1 64-bit x86 Dual Memory Channel

The maximum available memory bandwidth was measured on the test system which was composed of an Intel Core i7-3770k processor operating at 3.5 GHz (without the use of "turbo boost"), 4 cores, 8 threads, 8 MB of shared cache memory, and 16 GB of dual channel DDR3-1600 memory (CL 9-9-9-27).

3.6.1.1 STREAM Benchmark

The manufacturer specifies a maximum possible bandwidth of 25.6 GB/s for this system and the results obtained using our modified `stream-inlift` benchmark for it are shown in Table 3.1, from where it can be seen that the maximum bandwidth attained irrespective of the kernel or the number of threads is $B = 22.0$ GB/s, which will be used as the normalizing factor for determining the bandwidth saturation index for all algorithms.

TABLE 3.1: Intel stream-inlift benchmark results (GB/s)

	Threads							
Kernel	1	2	3	4	5	6	7	8
Copy	20.1	22.0	21.9	21.7	21.7	21.6	21.6	21.5
Scale	19.9	21.7	21.7	21.7	21.1	21.5	21.5	21.4
Add	19.3	21.7	21.7	21.5	20.7	21.4	21.4	21.3
Triad	19.4	21.7	21.8	21.7	20.8	21.5	21.5	21.4
Inlift	19.6	21.7	21.7	21.6	20.9	21.4	21.3	21.3
Lift	18.7	21.7	21.8	21.7	20.6	21.5	21.4	21.3

3.6.1.2 Bandwidth Saturation Index

The bandwidth saturation index for the lnaive algorithm (which uses 5 MOPs) is given by

$$S_l = \frac{5whc}{tB} \tag{3.3}$$

where w is the width, h is the height, c is the number of bytes of each coefficient, and t is the measured running time for the lnaive algorithm in nano seconds (ns). Table 3.2 shows the bandwidth saturation index S_l for our vector forward lnaive implementation of the ibior5x3 DWT for various image sizes ($c = 2$ and $B = 22.0$ GB/s) from where it can be seen that it is capable of saturating the memory bandwidth ($S_l \geq 0.95$) with only 2 threads.

TABLE 3.2: Intel lnaive ibior5x3 bandwidth saturation index - S_l

	Threads							
Size	1	2	3	4	5	6	7	8
3840×2160	0.86	0.94	0.95	0.94	0.92	0.91	0.90	0.90
5760×3240	0.86	0.95	0.96	0.97	0.93	0.91	0.91	0.92
7680×4320	0.83	0.96	0.97	0.96	0.94	0.91	0.91	0.93
11520×6480	0.83	0.98	0.98	0.97	0.92	0.90	0.91	0.95
15360×8640	0.83	0.98	0.98	0.97	0.88	0.90	0.90	0.95

Given that the paraline algorithm only uses the minimum number of 2 MOPs, its bandwidth saturation index S_p, also known as the absolute saturation index, is given by

$$S_p = \frac{2whc}{tB} \tag{3.4}$$

and Table 3.3 shows the bandwidth saturation index S_p achieved by our vector forward paraline implementation of the ibior5x3 DWT for various image sizes ($c = 2$ and $B = 22.0$ GB/s) from where it can be seen that it is able to saturate the memory bandwidth ($S_p \geq 0.95$) with 4 threads.

TABLE 3.3: Intel paraline ibior5x3 bandwidth saturation index - S_p

	Threads							
Size	1	2	3	4	5	6	7	8
3840×2160	0.53	0.74	0.87	0.95	0.96	0.96	0.97	0.96
5760×3240	0.53	0.78	0.90	0.98	0.97	0.98	0.98	0.98
7680×4320	0.53	0.79	0.92	0.99	0.98	0.99	0.99	0.98
11520×6480	0.53	0.81	0.92	0.99	0.98	0.98	0.97	0.97
15360×8640	0.52	0.81	0.93	1.00	0.98	0.98	0.97	0.97

Given that S_p only accounts for 2 MOPs, **our vector paraline implementation was able to achieve maximum performance (minimum time) with 4 cores, which is the best possible for this system**.

3.6.1.3 Performance Counters

In fact, using the processor's performance counters [69, 72], it can be verified that the paraline algorithm has a much lower cache-miss ratio than the lnaive algorithm and a correspondingly higher number of instructions per cycle, as shown in Table 3.4 for the vector ibior5x3 (forward and reverse) DWT using 8 threads, 7680 × 4320 images, and 100 repetitions.

TABLE 3.4: Intel performane counters - ibior5x3 - 8 threads - 7680 × 4320 - 100 reps

lnaive			paraline		
222,571,291	cache-misses	# 39.138 % of all cache refs	61,985,044	cache-misses	# 12.928 %of all cache refs
568,689,571	cache-references		479,454,129	cache-references	
88,868,669,688	cycles		36,332,107,370	cycles	
30,387,508,026	instructions	# 0.34 insn per cycle	30,844,322,428	instructions	# 0.85 insn per cycle
3.596537554	seconds	time elapsed	1.661460676	seconds	time elapsed

It should be emphasized that the best possible implementation, using the minimal amount of system resources, can only hope to achieve a bandwidth saturation index $S_p = 1$ because this index only accounts for 2 MOPs (1 read and 1 write), which is the absolute minimum for any implementation.

3.6.2 32-bit ARM SBC

As stated before, the ARM architecture presents a more relaxed memory model in comparison to the Intel architecture and requires a proper implementation for the memory acquire and release semantics. Algorithm 16, which uses the `stdatomic.h` API available in the `C11` standard, should be used in substitution for Algorithm 1 and implements the correct memory access functions necessary for the correct functioning of the multi-threaded algorithms presented before. In fact, Algorithm 16 is general and can be used with any architecture with either weak or strong memory ordering.

The RPi3B uses a Broadcom BCM2837 SoC (quad core ARM Cortex-A53 with 512KB of L2 cache) operating at 1.3 GHz with 1 GB of DDR2 RAM (32-bit single channel bus) at 500 MHz (overclocked values).

Algorithm 16 ARM Macros and Definitions

```
#include <stdlib.h>
#include <stdatomic.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_num_threads() 1
#endif

typedef short coeff_t;
typedef char atom;
typedef atomic_char atomic;
#define atomic_load_acq(ptr) \
  atomic_load_explicit(ptr,memory_order_acquire)
#define atomic_store_rel(ptr,val) \
  atomic_store_explicit(ptr,val,memory_order_release)

inline static atom
__sync_val_cas(atomic *valptr,atom oldval,atom newval)
{
  atomic_compare_exchange_strong_explicit(valptr,&oldval,newval,
    memory_order_acq_rel,memory_order_acquire);
  return oldval;
}
#define CAS(valptr,oldval,newval) __sync_val_cas(valptr,oldval,newval)

#define NONE    0
#define BUSY    1
#define DONE    2
#define WORK    4
#define MORE    8

typedef struct shrState {
  int     height;
  int     ntypes;
  atomic  array[];
} shrState_t;

inline static atomic *
shrstate(shrState_t *ss,int type,int n)
{
  return &ss->array[((n * ss->ntypes) + type)];
}

#define _LINE   0
#define _COLUMN 1
#define lnstate(s,n) shrstate(s,_LINE,n)
#define clstate(s,n) shrstate(s,_COLUMN,n)

#define P(a,b)  ((((a) + (b)) + 1) >> 1) // predict
#define U(a,b)  ((((a) + (b)) + 2) >> 2) // update

// interleaved even sample
#define S(n)    s[((n) << 1)]
#define _S(n)   s[((n) << 1) * w]
// interleaved odd sample
#define D(n)    s[((n) << 1) + 1]
#define _D(n)   s[(((n) << 1) + 1) * w]
```

3.6.2.1 STREAM Benchmark

Due to the narrow memory bus interface, we have also included another stream benchmark kernel for this architecture, in addition to the new ones, called `incopy`, which does

a copy to the same address, i.e., reads and writes the same memory location. The results obtained using our modified `stream-inlift` benchmark for the RPi3B are shown in table 3.5.

TABLE 3.5: RPi3B stream-inlift benchmark results (GB/s)

		Threads			
Kernel	MOP	1	2	3	4
Incopy	2	2.43	2.25	2.13	2.17
Copy	3	4.48	4.15	3.72	3.66
Scale	3	3.35	4.01	3.56	3.51
Add	4	3.04	3.63	3.24	3.06
Triad	4	2.48	3.59	3.23	3.09
Inlift	4	1.48	2.23	2.30	2.21
Lift	5	1.89	3.01	3.03	2.88

From table 3.5 it can be seen that there is a wide variation of the available bandwidth depending on the kernel (memory access pattern) and the number of threads for the RPi3B. Closer inspection reveals that the RPi3B uses a single 256Mb×32 DDR2 memory chip operating at 500MHz (overclocked) and the numbers clearly show the bottleneck imposed by this architecture, which favors cost over performance, in relation to multiple simultaneous reads and writes, i.e., the actual available bandwidth decreases with the number of simultaneous read and write operations on the memory bus, reaching its maximum with a single thread and simple copy operations (single threaded "copy" kernel). This is in contrast with current desktop and server computers, which are able to use the maximum available bandwidth irrespective of the kernel and number of threads [73].

From the memory saturation point of view, this makes it very hard to actually compare different algorithms on this platform because they could have different memory access patterns and, therefore, different maximum available bandwidths. We can, however, find out if the current implementation is actually saturating the memory bus, given its memory pattern, by looking at the resulting run times for different number of threads.

However, we cannot state that the current algorithm reaches global minimum execution times on this platform because it cannot be guaranteed that there are not faster memory access patterns that could be used. Also, given the values in table 3.5, it could be the case that in order to actually maximize performance for this architecture, an implementation

would have to actually limit work, i.e., some threads should actually do nothing until a single threaded memory prefetcher serially brings the coefficients from the main memory into the cache.

3.6.2.2 Integer DWT

In order to have a better perspective of the RPi3B platform, we have also implemented vector versions of more complex integer DWTs.

The ibior13x7, also consisting of 1 predict and 1 update step but with twice the complexity of the ibior5x3 steps and 4 vanishing moments for both the low and high pass filters, achieves excellent results, comparable to the floating point CDF-9/7 transform, while maintaining perfect reversibility by being an integer transform and, like the ibior5x3, is non expansive with a DC gain of 1 and an AC gain of 2. The ibior13x7 is composed of the following predict and update steps, respectively

$$d_j^1 = d_j^0 - \left\lfloor \frac{9(s_j^0 + s_{j+1}^0) - (s_{j-1}^0 + s_{j+2}^0)}{16} + \frac{1}{2} \right\rfloor \tag{3.5}$$

$$s_j^1 = s_j^0 + \left\lfloor \frac{9(d_{j-1}^1 + d_j^1) - (d_{j-2}^1 + d_{j+1}^1)}{32} + \frac{1}{2} \right\rfloor \tag{3.6}$$

and its lnaive implementation uses 5 MOPs.

The icdf9x7, just like the CDF-9/7 floating point transform for which it is an approximation, has 2 predict and 2 update steps but no scaling (it has a DC gain of 1.25 and an AC gain of 1.6). Unlike the CDF-9/7 transform, it has only 2 vanishing moments for its low pass filter instead of the 4 vanishing moments of the CDF-9/7. It is also reversible and achieves good results even though it is an expansive transform and is given by the following successive predict and update steps

$$d_j^1 = d_j^0 - \left\lfloor \frac{3(s_j^0 + s_{j+1}^0)}{2} + \frac{1}{2} \right\rfloor \tag{3.7}$$

$$s_j^1 = s_j^0 - \left\lfloor \frac{(d_{j-1}^1 + d_j^1)}{16} + \frac{1}{2} \right\rfloor \tag{3.8}$$

$$d_j^2 = d_j^1 + \left\lfloor \frac{4(s_j^1 + s_{j+1}^1)}{5} + \frac{1}{2} \right\rfloor \tag{3.9}$$

$$s_j^2 = s_j^1 + \left\lfloor \frac{15(d_{j-1}^2 + d_j^2)}{32} + \frac{1}{2} \right\rfloor \tag{3.10}$$

and its lnaive implementation uses 8 MOPs.

Table 3.6 shows the running times in μs for the lnaive ibior5x3, ibior13x7, and icdf9x7 for the RPi3B from 1 to 4 threads.

TABLE 3.6: RPi3B lnaive time (μs)

	ibior5x3 (5 MOPs)				ibior13x7 (5 MOPs)				icdf9x7 (8 MOPs)			
Size	1	2	3	4	1	2	3	4	1	2	3	4
960×540	2306	2643	2651	2727	3361	2780	2762	2865	4554	4386	4305	4433
1280×720	4393	4513	4530	4705	6146	4697	4645	4777	8360	7262	7131	7372
1920×1080	9643	9550	9512	9861	14297	10130	9894	9984	18500	15537	15097	15465
2880×1620	21171	20954	20962	21316	33563	22454	21593	21513	41428	33409	33249	33599
3840×2160	36741	36808	36601	37088	59988	39147	37321	37283	72609	58989	58354	59063

From table 3.6, it is easy to see that our lnaive implementation of the ibior5x3 on the RPi3B easily saturates the memory bus for a single thread. As the lnaive algorithm uses 5 MOPs, setting $S_l = 1$ and $c = 2$ in 3.3 results in $B = 2.27$ GB/s using the 3840×2160 resolution and 3 threads, which equals within a margin of error the value of $B = 2.30$ GB/s obtained with our stream-inlift benchmark for the "inlift" kernel with 3 threads given in table 3.5. Therefore, the value $B = 2.30$ GB/s will be used as the maximum bandwidth when evaluating the lnaive, pipeline, and paraline algorithms.

It can also be seen that the running times for ibior5x3 and ibior13x7 converge to the same value, even though the complexity of the ibior13x7 is twice that of the ibior5x3. In fact, our lnaive implementation running times for the shown integer DWTs saturate the memory bus and converge to the same value per MOP.

Table 3.7 shows the running times in μs for the paraline ibior5x3, ibior13x7, and icdf9x7 for the RPi3B from 1 to 4 threads.

TABLE 3.7: RPi3B paraline time (μs)

	ibior5x3 (2 MOPs)				ibior13x7 (2 MOPs)				icdf9x7 (2 MOPs)			
Size	1	2	3	4	1	2	3	4	1	2	3	4
960×540	1313	1052	1032	1044	2652	1623	1213	1129	3160	1904	1410	1229
1280×720	2490	1779	1770	1680	4787	2724	1954	1746	5662	3145	2282	1997
1920×1080	5614	3946	3845	3834	10778	5807	4124	3808	12893	6735	4758	4145
2880×1620	12858	8563	8502	8528	24183	12783	8966	8455	29085	14698	10372	9094
3840×2160	22870	15213	15071	15084	43623	22561	16026	15174	51926	26268	18239	16015

The validity of our memory cost model can be assessed by noticing that the timings for all 3 DWTs converge to the same value as the number of threads increase, as expected, once the paraline algorithm only uses 2 MOPs, irrespective of the DWT complexity. Also, using tables 3.6 and 3.7, it can be observed that the times for the paraline algorithm approach a MOP ratio of 2.5 for the ibior5x3 and ibior13x7 and a MOP ratio of 4 for the icdf9x7, also as expected from our memory model.

3.6.2.3 Performance Counters

Just like the case for the Intel architecture, it can be verified that, using the processor's performance counters [69, 72], the paraline algorithm has a much lower cache-miss ratio than the lnaive algorithm and a correspondingly higher number of instructions per cycle, as shown in Table 3.8 for the vector ibior5x3 (forward and reverse) DWT using 4 threads, 3840×2160 images, and 100 repetitions.

TABLE 3.8: RPi3B Performane counters - ibior5x3 - 4 threads - 3840×2160 - 100 reps

lnaive			paraline		
38,609,828,277	cycles		16,676,032,484	cycles	
4,754,753,027	instructions	# 0.12 insn per cycle	4,695,117,356	instructions	# 0.28 insn per cycle
413,407,053	LLC-loads		266,577,309	LLC-loads	
167,659,472	LLC-load-misses	# 40.56% of all LL-cache hits	54,382,688	LLC-load-misses	# 20.40% of all LL-cache hits
413,407,053	LLC-stores		266,577,309	LLC-stores	
167,659,472	LLC-store-misses		54,382,688	LLC-store-misses	
8.320214788	seconds	time elapsed	3.713646249	seconds	time elapsed

3.6.2.4 Floating Point DWT

The popular Cohen-Daubechies-Feauveau CDF-9/7 (cdf9x7) floating point DWT was also implemented and benchmarked on the RPi3B. Its lifting equations are given by the following successive predict and update steps and final scaling (the scaling value used implies equal low pass band and high pass band gains of $\sqrt{2}$).

$$d_j^1 = d_j^0 + \alpha \left(s_j^0 + s_{j+1}^0 \right) \tag{3.11}$$

$$s_j^1 = s_j^0 + \beta \left(d_{j-1}^1 + d_j^1 \right) \tag{3.12}$$

$$d_j^2 = d_j^1 + \gamma \left(s_j^1 + s_{j+1}^1 \right) \tag{3.13}$$

$$s_j^2 = s_j^1 + \delta \left(d_{j-1}^2 + d_j^2 \right) \tag{3.14}$$

$$d_j^3 = \frac{d_j^2}{\kappa} \tag{3.15}$$

$$s_j^3 = \kappa s_j^2 \tag{3.16}$$

where

$$\begin{aligned} \alpha &\approx -1.586134342 \quad \beta \approx -0.05298011857 \\ \gamma &\approx 0.8829110755 \quad \delta \approx 0.4435068520 \end{aligned} \tag{3.17}$$

and

$$\kappa \approx 1.149604399 \tag{3.18}$$

Its implementation uses 32-bit floating point coefficients which means that it is expected that, if memory saturation is achieved, its calculation should be 2 times slower than the previous 16-bit implementations because it uses twice as much memory.

As can be seen on tables 3.9 and 3.10 for the lnaive and paraline implementations, respectively, it is clear that our implementation easily saturates the memory bus with 2 threads for the lnaive algorithm, which uses 9 MOPS, and saturates the memory bus using 4 threads for the paraline implementation, also achieving minimal running time on this platform for this DWT.

TABLE 3.9: RPi3B lnaive cdf9x7 (9 MOPs) time (μs)

	Threads			
Size	1	2	3	4
960×540	9779	8992	8938	9215
1280×720	17113	15124	15141	15631
1920×1080	38092	33181	33035	33878
2880×1620	90588	73710	74081	75413
3840×2160	159982	128727	130791	132675

TABLE 3.10: RPi3B paraline cdf9x7 (2 MOPs) time (μs)

	Threads			
Size	1	2	3	4
960×540	5538	3119	2317	2058
1280×720	10119	5477	3977	3553
1920×1080	22477	12537	8830	7741
2880×1620	50169	27630	19521	17366
3840×2160	89039	49044	34368	32211

All algorithms described here work inplace, do not use any auxiliary memory, and maintain the interleaved state of the original matrix. In normal use, however, an N-level dyadic decomposition of a matrix requires that an originally interleaved matrix gets replaced by its deinterleaved transformation, which then has its low pass band serving as the next decomposition's interleaved matrix. In this scenario, all inplace transform algorithms will have to do a deinterleaving operation on the transformed matrix, for each decomposition level, which can be done inplace and requires an extra 2 memory operations and 1 line's worth of auxiliary memory.

3.7 Summary

The novel paraline algorithm for the calculation of the DWT based on the lifting scheme is presented. It works inplace without the need of any auxiliary memory and is able keep a serial memory access pattern and explore the cache efficiently in a multi threaded environment.

By using a low overhead thread synchronization scheme without the use of any locks, it was possible to reach memory saturation with the minimum number of memory accesses and, therefore, maximize performance for the test system.

Claims of linear scaling with the number of threads only apply in situations where the implementation is still far from saturating the memory bus. However, with the almost universal availability of vector instructions with their respective increase in efficiency, implementations are able to achieve saturation with a relatively small number of cores.

It was possible to achieve minimum latency (total memory saturation) for the ibior5x3 DWT using 4 threads on the x86_64 test system and using 2 threads on the ARM system. In practical terms this means that **a better or more efficient implementation could saturate the memory bus using less system resources but would not speed it up anymore, as it is not possible to do the calculations with less than 2 memory operations (1 read and 1 write)**.

An absolute metric based on the memory operations performed and on the available system memory bandwidth is introduced so that algorithms and their implementations can be compared. In this regard, the paraline algorithm is shown to be optimal and

its implementation offers a significant improvement over existing algorithms by reaching minimum latencies with a small number of cores and without the use of any auxiliary memory.

Chapter 4

Data Prioritization

This chapter address the central problem of code embedding, i.e., the selection of, among all possibilities, which data to send next, i.e., deals with data prioritization. More specifically, how to order the digits of the binary representation of a set of coefficients distributed according to a PDF $p(x)$ so that the distortion (MSE) of any prefix[1] of the resulting ordered stream is minimized. It is important to note that the work presented here does not change the contents but only the order in which data is encoded.

To avoid confusion, the upper case word "BIT" will be used as an acronym for "**B**inary dig**IT**", denoting the information to be encoded (either a '0' or a '1'), while the lowercase word "bit" will be used to denote the unit of binary entropy required to encode such information (a real number).

The best embedding order is achieved by, among all possibilities at any point, appending the information with the highest distortion reduction per bit. Generally speaking, if bit level embedding is not a requirement, there may be better compression schemes that may relay information regarding values which do not contribute for the distortion reduction at this point but may help with compression at a later stage, achieving higher compression rates in the end. This is specially true for non embedded algorithms, which only target the final size, or by block oriented embedded algorithms, which are still progressive but not at the bit level.

[1] at certain bit boundaries

The value of the distortion reduction per bit (ΔD/bit) is derived by first finding out the value of the distortion reduction per BIT (ΔD/BIT) and then dividing it by its respective entropy per BIT (bit/BIT).

Formulas for the distortion reduction per BIT along with the entropy per BIT for both significant and refinement BITs are derived for a general PDF $p(x)$ and an analysis is done for the particular case where $p(x)$ is an Exponential Power Distribution (EPD). Closed formulas for the uniform and laplace distributions, which can be viewed as special cases of the EPD, are shown along with a brief analysis of some of their special properties.

4.1 Introduction

Without loss of generality, the coefficients are assumed to be integer coefficients. Finite precision real coefficients may be appropriately scaled and subsequently rounded or truncated.

Each coefficient has a sign ('+' or '−'), and a magnitude $c \geq 0$ (coefficients where $c = 0$ have no sign) which implies that the binary representation of this coefficient has $\lceil \log_2(c+1) \rceil$ BITs, where $\lceil x \rceil$ represents the smallest integer greater than or equal to x (ceil).

A coefficient is said to be significant or test positive for significance at bitplane $p \geq 0$ iff its magnitude $c \in [2^p, 2^{p+1})$ or, equivalently, $p = \lfloor \log_2(c) \rfloor$, where $c > 0$ and $\lfloor x \rfloor$ represents the largest integer smaller than or equal to x (floor). In general, a coefficient is said to be significant at a positive interval $[a, b)$ whenever its magnitude $c \in [a, b)$, i.e., $0 < a \leq c < b$. The remaining p BITs $(p-1, \ldots, 1, 0)$ of a coefficient are called refinement BITs.

We define $w_{s,i}$ as being the amount of distortion reduction achieved by encoding the value of the i-th BIT of a coefficient that is significant at bitplane s. It should be observed that $0 \leq i \leq s$ so that $w_{s,s}$ represents the distortion reduction for the significant BIT, $w_{s,s-1}$ represents the distortion reduction for the first refinement BIT, $w_{s,s-2}$ represents the distortion reduction for the second refinement BIT and so on up to $w_{s,0}$, which represents the distortion reduction for the last refinement BIT of coefficients with magnitudes $c \in [2^s, 2^{s+1})$.

Defining the maximum number of bitplanes m as the number of bitplanes necessary to represent the maximum magnitude among all coefficients in an initial set, there are $m(m+1)/2$ values of w that need to be defined and sorted. Of these, m are due to significant BITs and $m(m-1)/2$ are due to refinement BITs.

Table 4.1 shows the 45 weights when the maximum number of bitplanes is 9. Generally speaking, the expected or "natural" order of decreasing distortion reduction values is from right to left (higher to lower bitplane), top to bottom (significance followed by increasing refinement levels). For example, in the depicted 9 bitplanes case, the first 10 weights in decreasing "natural" order are $w_{8,8}$, $w_{7,7}$, $w_{8,7}$, $w_{6,6}$, $w_{7,6}$, $w_{8,6}$, $w_{5,5}$, $w_{6,5}$, $w_{7,5}$, and $w_{8,5}$.

TABLE 4.1: Distortion Reduction values for 9 bitplanes

$[2^0, 2^1)$	$w_{0,0}$								
$[2^1, 2^2)$	$w_{1,0}$	$w_{1,1}$							
$[2^2, 2^3)$	$w_{2,0}$	$w_{2,1}$	$w_{2,2}$						
$[2^3, 2^4)$	$w_{3,0}$	$w_{3,1}$	$w_{3,2}$	$w_{3,3}$					
$[2^4, 2^5)$	$w_{4,0}$	$w_{4,1}$	$w_{4,2}$	$w_{4,3}$	$w_{4,4}$				
$[2^5, 2^6)$	$w_{5,0}$	$w_{5,1}$	$w_{5,2}$	$w_{5,3}$	$w_{5,4}$	$w_{5,5}$			
$[2^6, 2^7)$	$w_{6,0}$	$w_{6,1}$	$w_{6,2}$	$w_{6,3}$	$w_{6,4}$	$w_{6,5}$	$w_{6,6}$		
$[2^7, 2^8)$	$w_{7,0}$	$w_{7,1}$	$w_{7,2}$	$w_{7,3}$	$w_{7,4}$	$w_{7,5}$	$w_{7,6}$	$w_{7,7}$	
$[2^8, 2^9)$	$w_{8,0}$	$w_{8,1}$	$w_{8,2}$	$w_{8,3}$	$w_{8,4}$	$w_{8,5}$	$w_{8,6}$	$w_{8,7}$	$w_{8,8}$
	0	1	2	3	4	5	6	7	8

bitplane

It is quite clear that, for every row in table 4.1, which is equivalent to the set of coefficients that are significant at the interval $[2^s, 2^{s+1})$, the significant distortion reduction value must be greater than the refinement distortion reduction values to the left of it, i.e., $w_{s,i} < w_{s,s}$ for $0 \leq i < s$ and, even though the refinement distortion reduction values do not necessarily have to decrease from right to left on the same line, it is usually enforced that $w_{s,j} < w_{s,i}$ for $0 \leq j < i < s$.

As will be shown, when the PDF $p(x)$ is either an uniform or a laplace distribution, the refinement distortion reduction values for each bitplane are equal, i.e., $w_{i,k} = w_{j,k}$, $k < i < j < m$. In these cases, the number of distinct distortion reduction values in table 4.1 is reduced to $2m-1$ values, shown in table 4.2, of which m are due to significant bits and $m-1$ are due to refinement bits.

TABLE 4.2: Uniform/Laplace Distortion Reduction values for 9 bitplanes

$[2^0, 2^1)$	$w_{0,0}$								
$[2^1, 2^2)$	$w_{r,0}$	$w_{1,1}$							
$[2^2, 2^3)$	$w_{r,0}$	$w_{r,1}$	$w_{2,2}$						
$[2^3, 2^4)$	$w_{r,0}$	$w_{r,1}$	$w_{r,2}$	$w_{3,3}$					
$[2^4, 2^5)$	$w_{r,0}$	$w_{r,1}$	$w_{r,2}$	$w_{r,3}$	$w_{4,4}$				
$[2^5, 2^6)$	$w_{r,0}$	$w_{r,1}$	$w_{r,2}$	$w_{r,3}$	$w_{r,4}$	$w_{5,5}$			
$[2^6, 2^7)$	$w_{r,0}$	$w_{r,1}$	$w_{r,2}$	$w_{r,3}$	$w_{r,4}$	$w_{r,5}$	$w_{6,6}$		
$[2^7, 2^8)$	$w_{r,0}$	$w_{r,1}$	$w_{r,2}$	$w_{r,3}$	$w_{r,4}$	$w_{r,5}$	$w_{r,6}$	$w_{7,7}$	
$[2^8, 2^9)$	$w_{r,0}$	$w_{r,1}$	$w_{r,2}$	$w_{r,3}$	$w_{r,4}$	$w_{r,5}$	$w_{r,6}$	$w_{r,7}$	$w_{8,8}$
	0	1	2	3	4	5	6	7	8
	bitplane								

4.2 Probability of an Interval

In order to simplify notation, the probability of coefficients distributed according to a continuous PDF $p(x)$ being in the k-th sub-interval of length $(b-a)/2^n$ of the interval $[a, b)$ is given by

$$\mathcal{P}_{n,k} = \int_{a+k\delta_n}^{a+(k+1)\delta_n} p(x)dx, \quad 0 \leq k < 2^n \tag{4.1}$$

where

$$\delta_n = \frac{b-a}{2^n} \tag{4.2}$$

so that

$$\mathcal{P}_{0,0} = \int_a^b p(x)dx = \sum_{k=0}^{2^n-1} \mathcal{P}_{n,k}, \quad n = 0, 1, 2, \ldots \tag{4.3}$$

4.3 Distortion

The distortion $\mathbb{D}$ produced by coefficients in the interval $[a, b)$, assuming that the reconstruction value in this interval is $\hat{x}$, is given by

$$\mathbb{D} = \frac{1}{\mathcal{P}_{0,0}} \int_a^b (\hat{x} - x)^2 p(x)dx \tag{4.4}$$

where the factor $1/\mathcal{P}_{0,0}$ is due to the fact that we are interested in the distortion conditioned to the coefficients being significant in the interval $[a, b)$. In fact, the information that coefficients are not significant in this interval does not contribute to any reduction in distortion but only affects the entropy of the significance information. Also, the

refinement information of a coefficient is always encoded with prior knowledge of its significance information, i.e., is also conditioned to $x \in [a, b)$.

4.4 Reconstruction Value

The reconstruction value that minimizes the distortion $\mathbb{D}$ is found by solving for the derivative of $\mathbb{D}$ in relation to $\hat{x}$, i.e., $\mathbb{D}'(\hat{x}) = 0$ so that

$$\frac{d\mathbb{D}}{d\hat{x}} = \frac{2}{\mathcal{P}_{0,0}} \int_a^b (\hat{x} - x) p(x) dx = 0 \tag{4.5}$$

which, after solving for $\hat{x}$, yields the well known result that the reconstruction value that minimizes the distortion (MSE) of a random variable distributed according to a PDF $p(x)$ in an interval $[a, b)$ is the centroid of $p(x)$ in this interval, i.e.,

$$\hat{x} = \frac{\int_a^b x p(x) dx}{\int_a^b p(x) dx} \tag{4.6}$$

4.5 Distortion Reduction per BIT

4.5.1 Significant BIT

The distortion reduction achieved by encoding the information that coefficients are actually significant in the interval $[a, b)$ is, therefore, the difference between the distortion before, when the reconstruction value was 0, and after, when the reconstruction value is the centroid $\hat{x}$ of this interval, so that

$$\begin{aligned} \Delta\mathbb{D} &= \frac{1}{\mathcal{P}_{0,0}} \int_a^b \left[(0 - x)^2 - (\hat{x} - x)^2 \right] p(x) dx \\ &= 2\hat{x} \frac{\int_a^b x p(x) dx}{\int_a^b p(x) dx} - \hat{x}^2 \\ &= \hat{x}^2 \end{aligned} \tag{4.7}$$

which is simply the squared value of the centroid of $p(x)$ in the interval $[a, b)$.

4.5.2 Refinement BIT

The first refinement BIT will encode the information whether the coefficient is in the first or second half of the interval $[a, b)$. Defining $h = (a + b)/2$ as the midpoint of this interval, the centroid of the interval $[a, h)$ is given by

$$\hat{x}_0 = \frac{\int_a^h xp(x)dx}{\int_a^h p(x)dx} \tag{4.8}$$

and the centroid of the interval $[h, b)$ is given by

$$\hat{x}_1 = \frac{\int_h^b xp(x)dx}{\int_h^b p(x)dx} \tag{4.9}$$

The distortion reduction achieved by encoding the first refinement BIT is, therefore, the difference in distortion before, when the reconstruction value was $\hat{x}$ and after, when it is either $\hat{x}_0$ or $\hat{x}_1$, depending on which half of the interval the coefficient is in, i.e.,

$$\begin{aligned}\Delta\mathbb{D}_1 &= \frac{1}{\mathcal{P}_{0,0}} \int_a^h \left[(\hat{x} - x)^2 - (\hat{x}_0 - x)^2\right] p(x)dx \\ &+ \frac{1}{\mathcal{P}_{0,0}} \int_h^b \left[(\hat{x} - x)^2 - (\hat{x}_1 - x)^2\right] p(x)dx \\ &= (\hat{x}_0 - \hat{x})^2 \frac{\mathcal{P}_{1,0}}{\mathcal{P}_{0,0}} + (\hat{x}_1 - \hat{x})^2 \frac{\mathcal{P}_{1,1}}{\mathcal{P}_{0,0}}\end{aligned} \tag{4.10}$$

Using a similar reasoning it can be shown that the distortion reduction achieved by coding the n-th refinement BIT is given by

$$\Delta\mathbb{D}_n = \sum_{k=0}^{2^{n-1}-1} \left[(\hat{x}_{k0} - \hat{x}_k)^2 \frac{\mathcal{P}_{n,2k}}{\mathcal{P}_{0,0}} + (\hat{x}_{k1} - \hat{x}_k)^2 \frac{\mathcal{P}_{n,2k+1}}{\mathcal{P}_{0,0}}\right], \quad n = 1, 2, \ldots \tag{4.11}$$

where $\delta_n = (b-a)/2^n$, $\hat{x}_k$ is the centroid of $p(x)$ in the interval $[a+(2k)\delta_n, a+(2k+2)\delta_n)$, $\hat{x}_{k0}$ is the centroid of $p(x)$ in the interval $[a + (2k)\delta_n, a + (2k + 1)\delta_n)$, and $\hat{x}_{k1}$ is the centroid of $p(x)$ in the interval $[a + (2k + 1)\delta_n, a + (2k + 2)\delta_n)$.

4.6 Entropy per BIT

4.6.1 Significant BIT

Assuming an even-symmetric PDF $p(x)$ with zero mean and that the significance information of all coefficients with magnitudes greater than or equal to b have already been encoded, the probability of a remaining coefficient being significant in an interval $[a, b)$, i.e., a coefficient whose magnitude c is such that $0 < a \leq c < b$, is given by the conditional probability $p_s = P(a \leq c \mid c < b)$ and, using the conditional probability identity

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{4.12}$$

it can be reduced to

$$p_s = P(a \leq c \mid c < b) = \frac{P(a \leq c < b)}{P(c < b)} \tag{4.13}$$

so that

$$p_s = \frac{\int_a^b p(x)dx}{\int_0^b p(x)dx} \tag{4.14}$$

It should be noted that the significant coefficients are, by definition, equally distributed among positive and negative coefficients and, therefore, the probability for each sign is $p_s/2$. The number of bits necessary to encode the significance information of a coefficient, either negative significant, insignificant, or positive significant, is given by the entropy

$$\begin{aligned} q_s &= H\left(\frac{p_s}{2}\right) + H\left(1 - p_s\right) + H\left(\frac{p_s}{2}\right) \\ &= p_s + H(p_s) + H(1 - p_s) \\ &= p_s + \mathcal{H}(p_s) \end{aligned} \tag{4.15}$$

where H is the entropy function (in bits) defined as $H(0) = 0$ and

$$H(x) = -x \log_2(x), \quad 0 < x \leq 1 \tag{4.16}$$

and $\mathcal{H}(x)$ is the binary entropy function, also in bits, defined as

$$\mathcal{H}(x) = H(x) + H(1 - x), \quad 0 \leq x \leq 1 \tag{4.17}$$

and plotted in figure 4.1 where it can be seen that it is an even symmetric function around the point $x = 0.5$, i.e., $\mathcal{H}(x) = \mathcal{H}(1-x)$ for $0 \leq x \leq 1$.

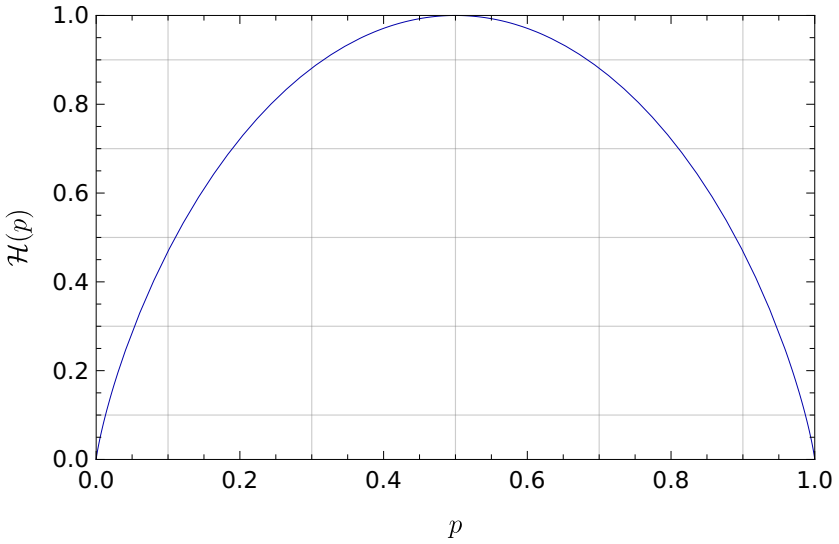


FIGURE 4.1: Binary Entropy $\mathcal{H}(p)$

The entropy per significant BIT is, therefore, given by the entropy of the significance information q_s (including sign and insignificant information) divided by the ratio of the number of significant coefficients in relation to the total number of coefficients (significant and insignificant), which is exactly the conditional probability of a significant coefficient p_s so that, using 4.15 and 4.14,

$$\eta = \frac{q_s}{p_s} = \frac{p_s + \mathcal{H}(p_s)}{p_s} = 1 + \frac{\mathcal{H}(p_s)}{p_s} \tag{4.18}$$

whose reciprocal $1/\eta$ is plotted in figure 4.2.

4.6.2 Refinement BIT

When encoding the n-th refinement level for coefficients which are significant in the interval $[a, b)$, it is assumed that all previous refinement levels $(1, 2, \cdots, n-1)$ from higher bitplanes have already been encoded. Also, refinement BITs always reduce distortion, irrespective of their value.

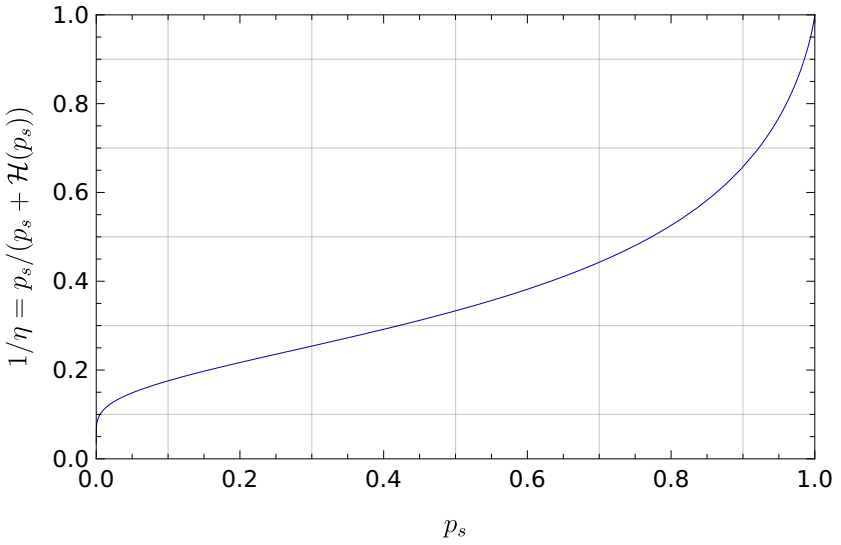


FIGURE 4.2: Reciprocal Significant bits/BIT

Most set partitioning algorithms simply encode the value of the refinement BIT (0 or 1) so that, in this case, the number of bits per refinement BIT is exactly 1, i.e.,

$$\eta_n = 1, \quad n = 1, 2, \ldots \tag{4.19}$$

which is a good approximation as long as the probability of each value of a refinement BIT (0 or 1) is close to 0.5.

Otherwise, a more rigorous analysis could be carried out for entropy coding algorithms. In this case, it is easy to see that every refinement BIT reveals if the coefficient is on the left or on the right half of the interval where it currently is so that the entropy of each refinement BIT is the binary entropy $\mathcal{H}$ of the conditional probability of the left or right half of the given current interval, which are the same, i.e., $\mathcal{P}_{n-1,k} = \mathcal{P}_{n,2k} + \mathcal{P}_{n,2k+1}$.

The final entropy for the n-th refinement level is, therefore, the expected value of the entropies of each of the 2^{n-1} sub-intervals, for coefficients which are significant in the interval $[a, b)$, which is given by the weighted sum

$$\eta_n = \sum_{k=0}^{2^{n-1}-1} \mathcal{H}\left(\frac{\mathcal{P}_{n,2k}}{\mathcal{P}_{n-1,k}}\right) \frac{\mathcal{P}_{n-1,k}}{\mathcal{P}_{0,0}}, \quad n = 1, 2, \ldots \tag{4.20}$$

4.7 Distortion Reduction per Bit

The desired value of the distortion reduction per bit for both significant and refinement BITs can be finally calculated by simply dividing each respective distortion reduction per BIT by its entropy per BIT (bits per BIT).

4.7.1 Significant BIT

The distortion reduction per significant bit $\Delta\mathcal{D}$ can be derived by dividing the distortion reduction per significant BIT $\Delta\mathbb{D}$ (4.7) by the entropy per significant BIT η (4.18) resulting in

$$\Delta\mathcal{D} = \frac{\Delta\mathbb{D}}{\eta} = \frac{\hat{x}^2}{1 + \mathcal{H}(p_s)/p_s} \tag{4.21}$$

remembering that the significant coefficients have their magnitudes constrained to be in the positive interval $[a, b)$ with $\hat{x}$ given in 4.6 and p_s given in 4.14.

4.7.2 Refinement BIT

Likewise, the distortion reduction per refinement bit $\Delta\mathcal{D}_n$ can be derived by dividing the distortion reduction per refinement BIT $\Delta\mathbb{D}_n$ (4.11) by the entropy per refinement BIT η_n resulting in

$$\Delta\mathcal{D}_n = \frac{\Delta\mathbb{D}_n}{\eta_n}, \quad n = 1, 2, \ldots \tag{4.22}$$

Most set partitioning algorithms simply convey the value of the refinement bit so that, in this case, the entropy per refinement BIT is exactly 1 (4.19) and the distortion reduction per refinement bit $\Delta\mathcal{D}_n$ is the same as the distortion reduction per refinement BIT $\Delta\mathbb{D}_n$ (4.11), i.e.,

$$\Delta\mathcal{D}_n = \frac{\Delta\mathbb{D}_n}{1} = \sum_{k=0}^{2^{n-1}-1} \left[(\hat{x}_{k0} - \hat{x}_k)^2 \frac{\mathcal{P}_{n,2k}}{\mathcal{P}_{0,0}} + (\hat{x}_{k1} - \hat{x}_k)^2 \frac{\mathcal{P}_{n,2k+1}}{\mathcal{P}_{0,0}} \right], \quad n = 1, 2, \ldots \tag{4.23}$$

For entropy coding algorithms, the distortion reduction per refinement bit $\Delta\mathcal{D}_n$ can be written by dividing $\Delta\mathbb{D}_n$ (4.11) by the real entropy of the refinement information η_n

(4.20) resulting in

$$\Delta\mathcal{D}_n = \frac{\sum_{k=0}^{2^{n-1}-1} \left[(\hat{x}_{k0} - \hat{x}_k)^2 \mathcal{P}_{n,2k} + (\hat{x}_{k1} - \hat{x}_k)^2 \mathcal{P}_{n,2k+1}\right]}{\sum_{k=0}^{2^{n-1}-1} \mathcal{H}\left(\frac{\mathcal{P}_{n,2k}}{\mathcal{P}_{n-1,k}}\right) \mathcal{P}_{n-1,k}}, \quad n = 1, 2, \ldots \tag{4.24}$$

4.8 The Exponential Power Distribution

The exponential power distribution (EPD), also known as the generalized normal distribution or generalized gaussian distribution, is defined as having zero mean $\mu = 0$, shape parameter s, and variance σ^2 by its PDF

$$g(x; s, \sigma^2) = \kappa e^{-|\varphi x|^s} \tag{4.25}$$

where the positive constants φ and κ are given by

$$\varphi = \sqrt{\frac{\Gamma\left(\frac{3}{s}, 0\right)}{\sigma^2 \Gamma\left(\frac{1}{s}, 0\right)}} \tag{4.26}$$

and

$$\kappa = \frac{\varphi s}{2\Gamma\left(\frac{1}{s}, 0\right)} \tag{4.27}$$

where $\Gamma(s, x)$ is the upper incomplete gamma function, which is defined for parameters $s > 0$ and $x \geq 0$ as

$$\Gamma(s, x) = \int_x^\infty t^{s-1} e^{-t} dt \tag{4.28}$$

Both the laplacian and gaussian distributions are special cases of the EPD when the shape parameter $s = 1$ and $s = 2$, respectively. The uniform distribution can also be viewed as the limit of the EPD when the shape parameter approaches infinity, i.e., $s \to \infty$.

Figure 4.3 shows a plot of the EPD for some values of the shape parameter s and with variance $\sigma^2 = 1$ (the x axis can be viewed as x/σ and the y axis as $g(x/\sigma; s, 1)$ for different values of σ^2).

Also, it is well known that the distribution of the AC wavelet coefficients of natural images is well approximated by these fat-tail distributions for small values of the shape

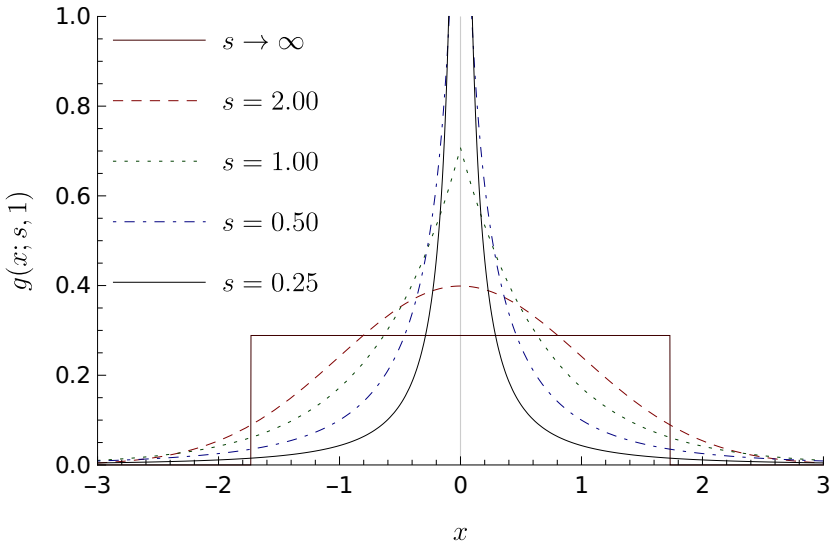


FIGURE 4.3: Exponential Power Distribution ($\sigma = 1$)

parameter s (values of $s = 0.5$ or even lower are not uncommon) which means that there are many small coefficient values but also a few large outliers [74].

It is easy to show that, for an EPD $p(x) = g(x; s, \sigma^2)$, an interval $[a, b)$ where $0 \leq a < b$, and an integer $n \geq 0$,

$$\int_a^b x^n p(x)dx = \frac{\Gamma\left(\frac{n+1}{s}, \alpha\right) - \Gamma\left(\frac{n+1}{s}, \beta\right)}{2\varphi^n \Gamma\left(\frac{1}{s}, 0\right)} \tag{4.29}$$

where

$$\alpha = (a\varphi)^s \quad \text{and} \quad \beta = (b\varphi)^s \tag{4.30}$$

and, using 4.6, 4.29, and 4.30, the centroid of an EPD $g(x; s, \sigma^2)$ can be written as

$$\hat{x} = \frac{1}{\varphi} \cdot \frac{\Gamma\left(\frac{2}{s}, \alpha\right) - \Gamma\left(\frac{2}{s}, \beta\right)}{\Gamma\left(\frac{1}{s}, \alpha\right) - \Gamma\left(\frac{1}{s}, \beta\right)} \tag{4.31}$$

4.8.1 Significant Entropy

Using 4.29 and 4.30, the conditional probability p_s (4.14) for an EPD can be rewritten as

$$p_s = \frac{\Gamma\left(\frac{1}{s}, \alpha\right) - \Gamma\left(\frac{1}{s}, \beta\right)}{\Gamma\left(\frac{1}{s}, 0\right) - \Gamma\left(\frac{1}{s}, \beta\right)} \tag{4.32}$$

so that the number of bits per significant BIT η (4.18) is log-linear plotted in figure 4.4 for various values of the shape parameter s and for binary intervals $[a, b) = [x/\sigma, 2x/\sigma)$, where σ is the standard deviation (square root of the variance σ^2).

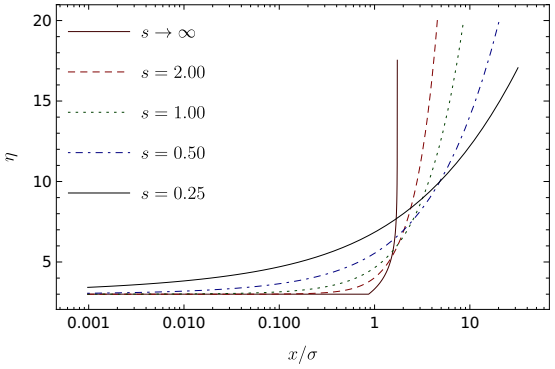


FIGURE 4.4: EPD η

4.8.2 Significant Distortion Reduction

Using 4.31, the relative square root of the distortion reduction per significant BIT $\Delta\mathbb{D}$ (4.7) and per significant bit $\Delta\mathcal{D}$ (4.21) are log-linear plotted in figures 4.5 and 4.6, respectively, for binary intervals $[a, b) = [x/\sigma, 2x/\sigma)$. The y axis shows the relative square root distortion reduction, i.e., $\sigma\sqrt{\Delta\mathbb{D}}/x$ and $\sigma\sqrt{\Delta\mathcal{D}}/x$, respectively.

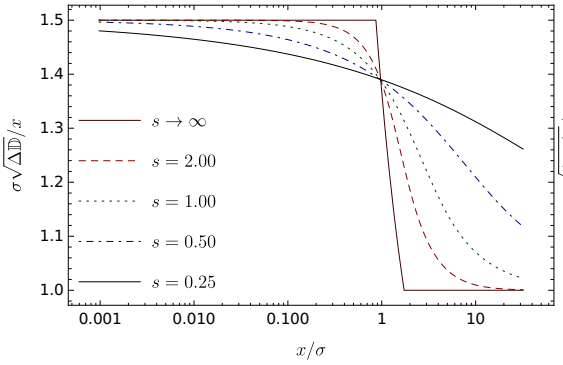


FIGURE 4.5: EPD $\Delta\mathbb{D}$

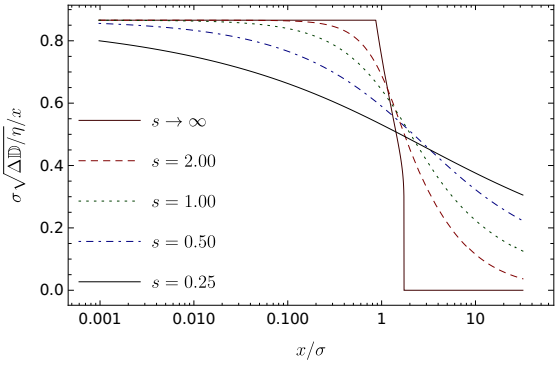


FIGURE 4.6: EPD $\Delta\mathcal{D}$

Comparing figures 4.5 and 4.6, it is quite clear that the distortion reduction per significant bit drops by a large factor specially for larger values of x, i.e., $x > \sigma$ which, when combined with a much smaller variation for the refinement bits which, as will be seen next, keep their relative values in the vicinity of 0.5, may eventually lead to some refinement levels to posses a higher distortion reduction per bit than significant levels at the same or higher bitplanes. This is specially true for small values of the shape

parameter s, when there are almost surely a number of "outliars" with high entropy requiring a large amount of bits to be encoded.

4.8.3 Refinement Entropy

For context-modeling entropy-coding algorithms which may use the actual entropy of the refinement information, the number of bits per refinement BIT η_n given in 4.20 is log-linear plotted for binary intervals $[a, b) = [x/\sigma, 2x/\sigma)$ for the first 3 refinement levels and shown in figures 4.7, 4.8, and 4.9.

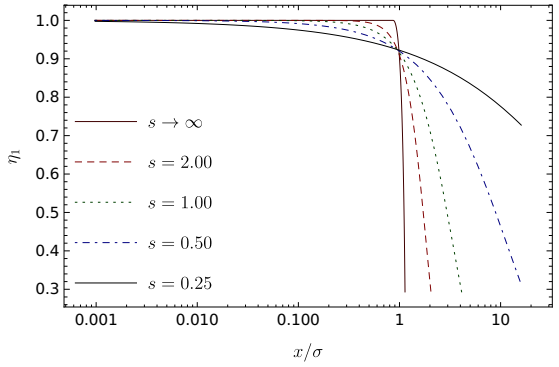


FIGURE 4.7: EPD η_1

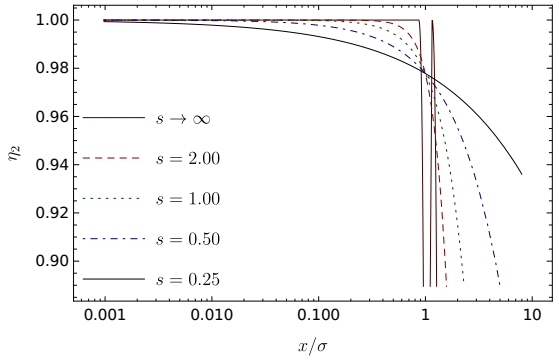


FIGURE 4.8: EPD η_2

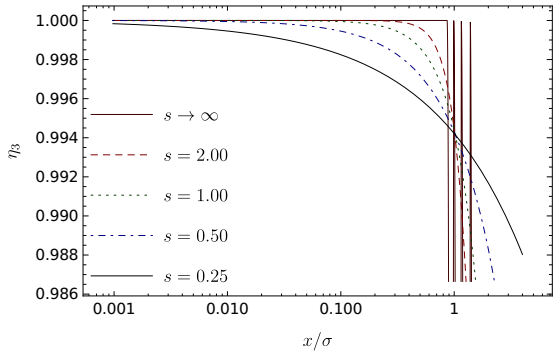


FIGURE 4.9: EPD η_3

It can be seen that while there certainly are gains for using entropy coding for the refinement coefficients these gains are mostly concentrated at the first few refinement levels, where the number of coefficients is small but, also being large in value, play an important role for very low bit rate compression and, therefore, good embedding.

4.8.4 Refinement Distortion Reduction

The first 3 levels of the scaled relative square root of the distortion reduction per refinement BIT $\Delta\mathbb{D}_n$ (4.11) and per refinement bit $\Delta\mathcal{D}_n$ (4.24), are log-linear plotted in figures 4.10, 4.12, and 4.14 and figures 4.11, 4.13, and 4.15, respectively, for binary intervals $[a, b) = [x/\sigma, 2x/\sigma)$. The y axis shows the scaled relative square root of the distortion reduction for the n-th refinement level $2^n\sigma\sqrt{\Delta\mathbb{D}_n}/x$ and $2^n\sigma\sqrt{\Delta\mathcal{D}_n}/x$, respectively.

It should be observed that the coefficients for the n-th refinement level are significant in the interval $[2^n x/\sigma, 2^{n+1}x/\sigma)$.

It can be seen that, for the case of set-partitioning algorithms where $\eta_n = 1$, $\Delta\mathcal{D}_n = \Delta\mathbb{D}_n$ so that figures 4.10, 4.12, and 4.14 show the scaled relative square root of the distortion reduction per refinement bit for the first 3 refinement levels.

A quick analysis of figures 4.10, 4.12, and 4.14 shows that the y values for a range of x values compatible with their respective shape parameter s does not deviate much from the uniform ($s \to \infty$) value of 0.5 and only do so by a small amount for the first few refinement levels. By being compatible with their respective shape parameter s we mean that the outliers will be larger for smaller values of s, e.g., for the first refinement level we won't find many values, if any, where $x \in [4\sigma, 8\sigma)$ range for a gaussian PDF ($s = 2$) but they will almost certainly exist in a PDF where $s = 0.5$, for example. It is also clear that as the refinement level gets higher (lower bitplanes) the values get closer to the ones of the uniform ($s \to \infty$) PDF.

Taking the real entropy η_n given in 4.20 into consideration and comparing the resulting figures of the EPD $\Delta\mathcal{D}_n$ with the respective ones of the EPD $\Delta\mathbb{D}_n$, it can be observed that the resulting distortion reduction gets even closer to the values of the uniform PDF ($s \to \infty$) by looking at the range of the values of the y axis.

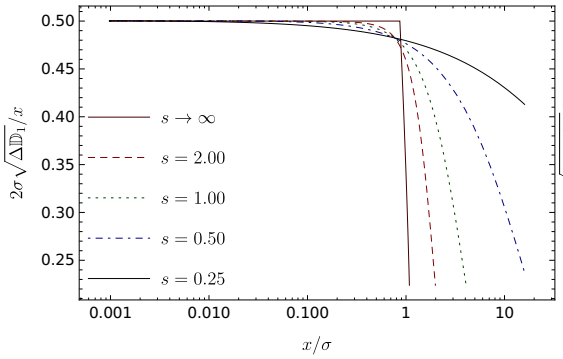


FIGURE 4.10: EPD $\Delta\mathbb{D}_1$

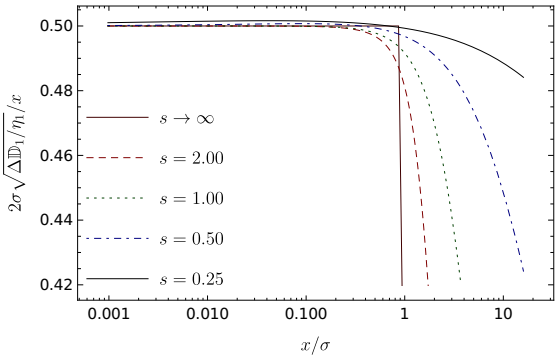


FIGURE 4.11: EPD $\Delta\mathcal{D}_1$

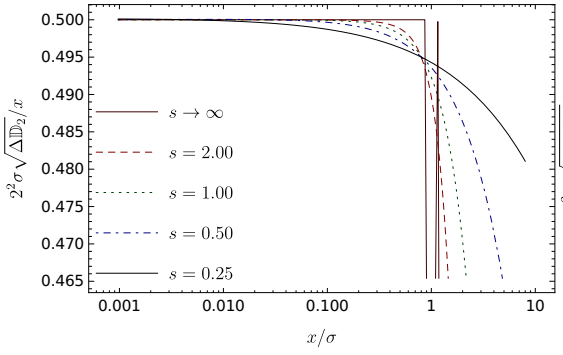


FIGURE 4.12: EPD $\Delta\mathbb{D}_2$

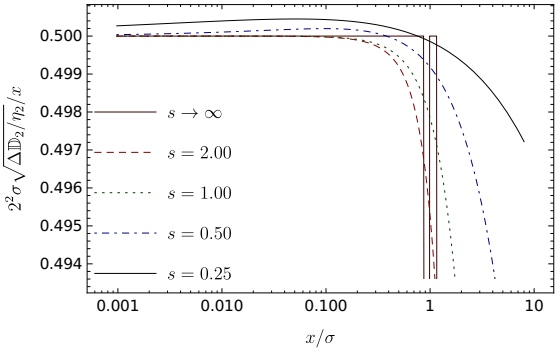


FIGURE 4.13: EPD $\Delta\mathcal{D}_2$

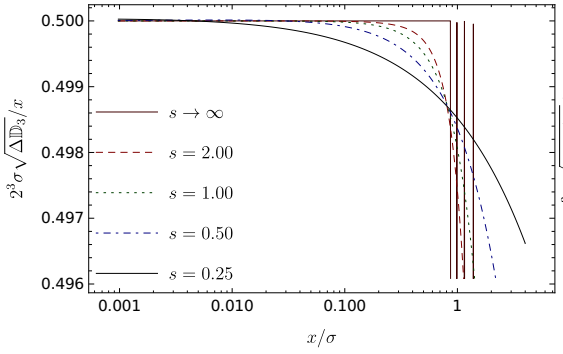


FIGURE 4.14: EPD $\Delta\mathbb{D}_3$

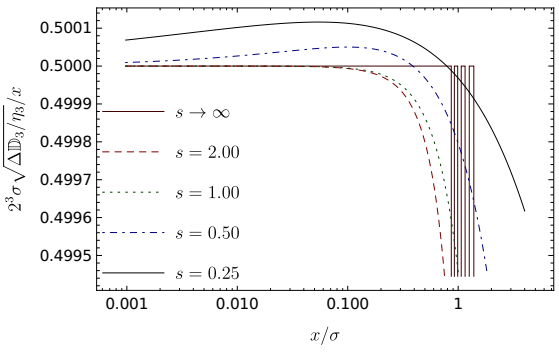


FIGURE 4.15: EPD $\Delta\mathcal{D}_3$

4.8.5 Distortion Reduction Ratio

In order to investigate the conditions when the distortion reduction due to refinement bits is higher than the one due to significant bits, figures 4.16, 4.17, 4.18, 4.19, and 4.20 show contour plots of the square root of the ratio $\Delta\mathcal{D}_n/\Delta\mathcal{D}$ for both cases when $\eta_n = 1$ and when η_n is given in 4.20 for the first 5 refinement levels, respectively. Each plot varies the shape parameter s from 0.1 to 1 (y axis) for binary intervals $[a, b) = [x/\sigma, 2x/\sigma)$ (x axis).

It should be made clear that for each refinement level n, the ratio given is for refinement

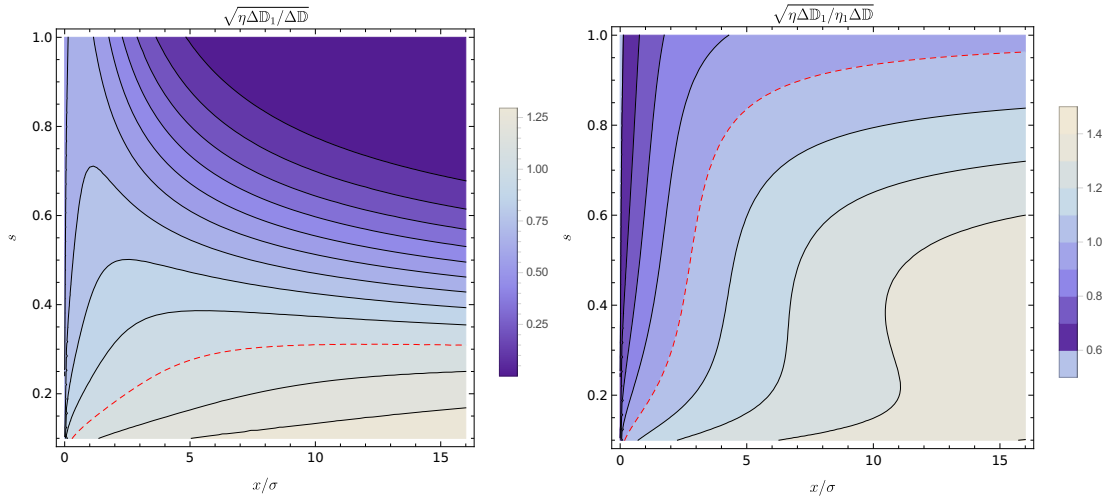


FIGURE 4.16: 1st refinement level distortion reduction ratio

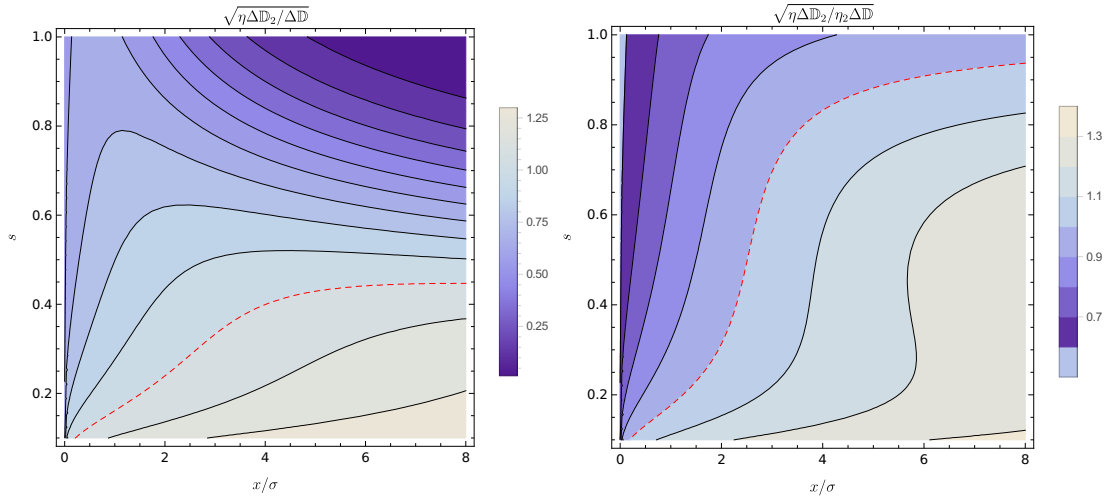


FIGURE 4.17: 2nd refinement level distortion reduction ratio

values in the range $[x/\sigma, 2x/\sigma)$ of coefficients that are significant at $[2^n x/\sigma, 2^{n+1}x/\sigma)$ divided by the significance values for coefficients which are significant at $[x/\sigma, 2x/\sigma)$.

In these figures, the dashed line represents where the ratio is 1, i.e., where both significant and refinement bits contribute the same amount of distortion reduction per bit and each adjacent line either decrease (left) or increase (right) the previous ratio by 0.1. Also, the first figure (left) of each pair shows the contour plot when $\eta_n = 1$ and the second (right) shows the ratio when η_n is given by 4.20.

From these figures it is quite clear that, as expected, the use of the real entropy for the refinement BITs (η_n given in 4.20) increases their contribution in the reduction of the distortion making the ratio $\Delta\mathcal{D}_n/\Delta\mathcal{D}$ achieve higher values for a wider range of shape

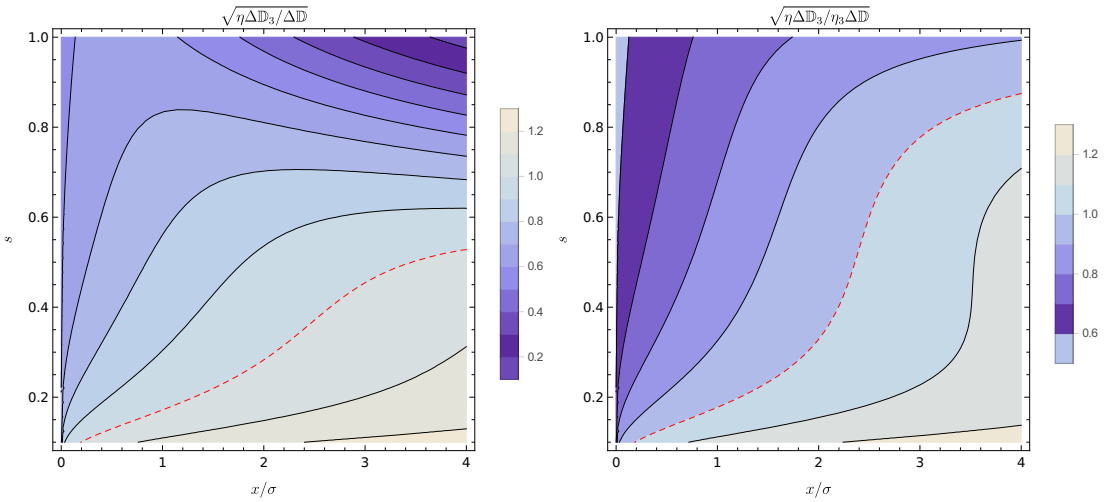


FIGURE 4.18: 3rd refinement level distortion reduction ratio

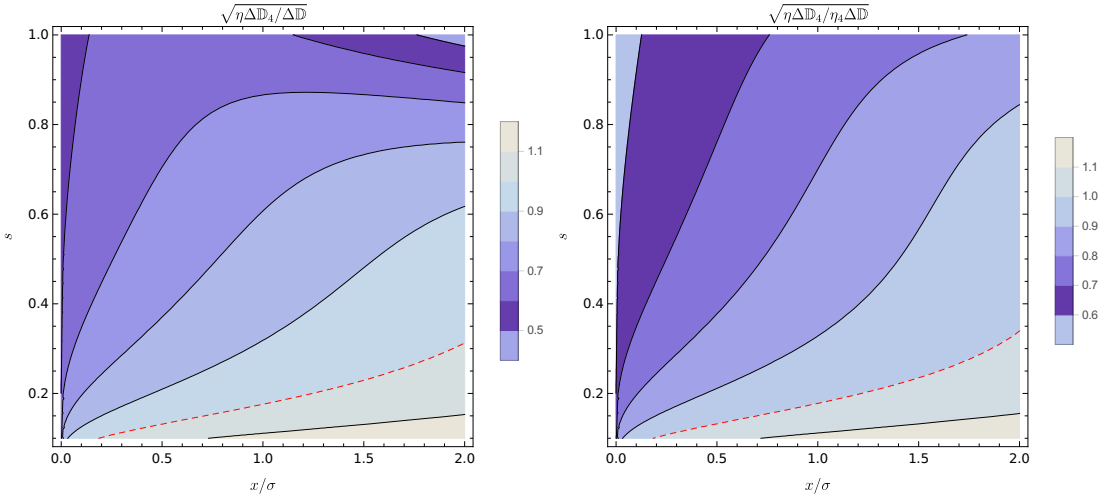


FIGURE 4.19: 4th refinement level distortion reduction ratio

parameters s and for lower ranges of x/σ for approximately the first 3 refinement levels, assuming that the shape parameter s of real images is usually in the range $[0.25, 0.75]$. As the refinement level increases (lower bitplanes), the differences between using or not the real entropy becomes less important and the contour plots for both start to look alike for the lower range of x/σ (it should be noted that the 5-th refinement level is for coefficients which are significant in the $[32x/\sigma, 64x/\sigma)$ range.

Also, it can be seen that, for the $\eta_n = 1$ case, both the second and third refinement levels achieve a ratio value of 1 at a shape parameter s greater than the first refinement level for values in corresponding binary intervals. From then on, the next refinement levels (lower bitplanes) seem to follow the expected behavior of a ratio smaller than one for most values in the expected x/σ range.

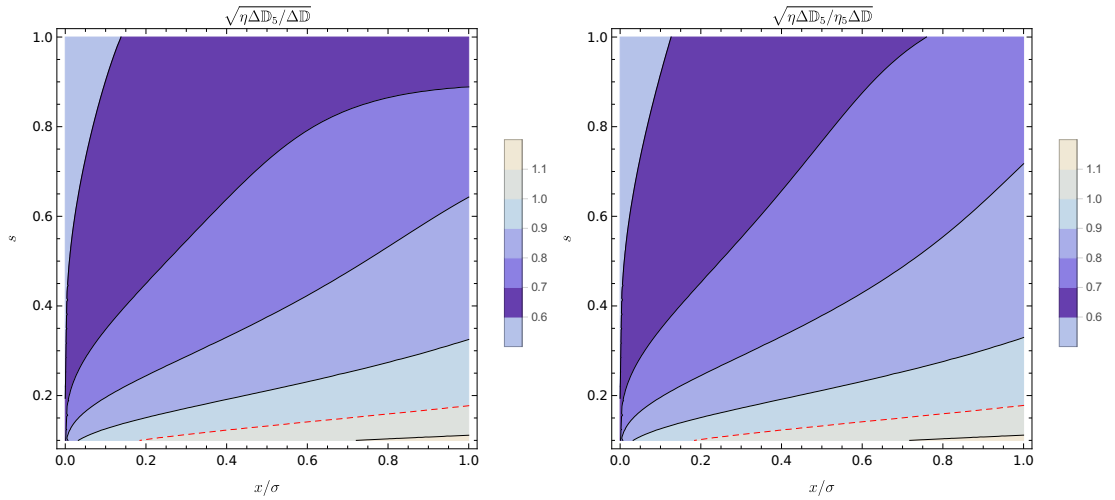


FIGURE 4.20: 5^{th} refinement level distortion reduction ratio

4.9 Uniform Distribution

For comparison purposes, the well known results for the uniform distribution are derived using the formulae given for the EPD by using the fact that the uniform distribution can be seen as the limit of an EPD when the shape parameter $s \to \infty$.

A uniform distribution with variance σ^2 is given by

$$p(x) = \begin{cases} \dfrac{1}{2\sqrt{3\sigma^2}}, & \text{if } |x| < \sqrt{3\sigma^2} \\ 0, & \text{otherwise} \end{cases} \tag{4.33}$$

It is easy to see from 4.6 that the centroid of any interval which is completely contained inside the limits of this distribution is the middle point of this interval. For simplicity, we assume that $\sqrt{3\sigma^2} = 2^N$ with N being a positive integer so that the last interval is also completely contained inside the limits.

Therefore, for binary intervals $[a, b) = [a, 2a)$, i.e., $b = 2a$, we have that, using 4.6, 4.14, and 4.18, $\hat{x} = 3a/2$, $p_s = 0.5$, and $\eta = 3$, respectively, so that from 4.21

$$\Delta\mathcal{D} = \frac{\Delta\mathbb{D}}{\eta} = \frac{\hat{x}^2}{\eta} = \left(\frac{\sqrt{3}}{2}a\right)^2 \tag{4.34}$$

It is also easy to see that $\eta_n = 1$ and, using 4.11,

$$\Delta\mathcal{D}_n = \frac{\Delta\mathbb{D}_n}{\eta_n} = \left(\frac{a}{2^{n+1}}\right)^2, \quad n = 1, 2, \ldots \tag{4.35}$$

Table 4.3 shows the values of the square root of the distortion reduction per bit for coefficients in binary intervals $[2^p, 2^{p+1})$, $0 \leq p < 9$, for an uniform distribution for the first 9 bitplanes under the assumption that $\sqrt{3\sigma^2} = 2^9$.

TABLE 4.3: Uniform root distortion reduction per bit ($\sqrt{3\sigma^2} = 2^9$)

0.866								
0.5	1.732							
0.5	1	3.464						
0.5	1	2	6.928					
0.5	1	2	4	13.856				
0.5	1	2	4	8	27.713			
0.5	1	2	4	8	16	55.426		
0.5	1	2	4	8	16	32	110.851	
0.5	1	2	4	8	16	32	64	221.703
0	1	2	3	4	5	6	7	8
bitplane								

It is clear that the values in table 4.3 follow the "natural" order of decreasing weights and also exhibit a single value for the refinement distortion reduction for each bitplane. These values, however, do not produce a sequence which is optimal in the rate-distortion sense for most natural images, which invariably posses different statistics. Nevertheless, its values are very simple to calculate and may be used when simplicity or speed is of paramount importance and are valued more than a precise rate-distortion optimized stream.

4.10 Laplace Distribution

In the special case of the laplace distribution (EPD with shape parameter $s = 1$), simpler closed form solutions can be derived for both the distortion reduction per significant BIT (4.7) and the distortion reduction per refinement BIT (4.11) by using the following recurrence property for the upper incomplete gamma function

$$\Gamma(n+1, x) = n!e^{-x}e_n(x), \quad n = 0, 1, 2, \ldots \tag{4.36}$$

where

$$e_n(x) = \sum_{k=0}^{n} \frac{x^k}{k!} \tag{4.37}$$

which, after some (quite extensive) algebraic manipulations, take on the following forms for $s = 1$ in the interval $[a, b)$

$$\Delta\mathbb{D} = \hat{x}^2 = \left[\frac{1}{\lambda} + a - \frac{b-a}{e^{\lambda(b-a)} - 1}\right]^2 \tag{4.38}$$

and

$$\Delta\mathbb{D}_n = \left[\frac{\delta_{n+1}}{\cosh(\lambda\delta_{n+1})}\right]^2, \quad n = 1, 2, \ldots \tag{4.39}$$

where

$$\lambda = \sqrt{\frac{2}{\sigma^2}}, \quad \delta_n = \frac{b-a}{2^n}, \quad \cosh(x) = \frac{e^x + e^{-x}}{2}$$

Close inspection of equation 4.39 reveals that $\Delta\mathbb{D}_n$ for an interval of length $(b-a)$ is the same as $\Delta\mathbb{D}_{n+1}$ for an interval of length $2(b-a)$, which is the same as $\Delta\mathbb{D}_{n+2}$ for an interval of length $2^2(b-a)$ and so on. Therefore, the distortion reduction per refinement BIT values are the same for each bitplane of the laplace distribution. Also, equation 4.38 tells us that the distance $(\hat{x} - a)$ is a function of the length of the interval $(b-a)$.

The normalized probability p_s given in 4.14 can, in this case, be rewritten as

$$p_s = \frac{e^{-\lambda a} - e^{-\lambda b}}{1 - e^{-\lambda b}} \tag{4.40}$$

and for binary intervals $[a, b) = [a, 2a)$ it can be further simplified resulting in

$$p_s = \frac{1}{1 + e^{\lambda a}} \tag{4.41}$$

so that

$$1 - p_s = e^{\lambda a} p_s \tag{4.42}$$

The number of bits per significant coefficient η given in 4.18 can be manipulated using equations 4.17, 4.41, and 4.42 resulting in

$$\begin{aligned}\eta &= 1 + \frac{\mathcal{H}(p_s)}{p_s} \\ &= 1 - \log_2(p_s) - \frac{1-p_s}{p_s}\log_2(1-p_s) \\ &= 1 - \frac{\lambda a e^{\lambda a}}{\log(2)} + (1+e^{\lambda a})\log_2(1+e^{\lambda a})\end{aligned} \tag{4.43}$$

In order to compare the weights of the laplacian and the uniform distributions, table 4.4 shows the values for a laplacian PDF with $\sigma^2 = 128^2$ for the first 9 bitplanes (the last interval is $[2\sigma, 4\sigma)$). It can be seen that there is very little change in relation to the uniform PDF except when the coefficients' magnitudes exceed the standard deviation (128 in this case).

TABLE 4.4: Laplace root distortion reduction per sig bit and ref BIT ($\sigma^2 = 128^2$)

0.864								
0.500	1.724							
0.500	1.000	3.430						
0.500	1.000	2.000	6.794					
0.500	1.000	2.000	3.996	13.324				
0.500	1.000	2.000	3.996	7.969	25.634			
0.500	1.000	2.000	3.996	7.969	15.753	47.508		
0.500	1.000	2.000	3.996	7.969	15.753	30.099	82.304	
0.500	1.000	2.000	3.996	7.969	15.753	30.099	50.770	128.961
0	1	2	3	4	5	6	7	8

bitplane

4.11 Shape parameter estimation

If it is known that a random variable X is distributed according to an exponential power distribution with zero mean $E(X) = \mu = 0$, its shape parameter s can be determined by using the ratio between its variance σ_X^2 and the square of its mean of absolute values $E^2(|X|)$ [75][76] by solving the following equation for s

$$r_v^2(s) = \frac{E\left(|X|^2\right)}{E^2\left(|X|\right)} = \frac{\sigma_X^2}{E^2\left(|X|\right)} = \frac{\Gamma\left(\frac{1}{s},0\right)\cdot\Gamma\left(\frac{3}{s},0\right)}{\Gamma^2\left(\frac{2}{s},0\right)} \tag{4.44}$$

or by solving the following equation relating the kurtosis [77] to the shape parameter of the exponential power distribution

$$r_k^2(s) = \frac{E\left(|X|^4\right)}{E^2\left(|X|^2\right)} = \frac{E\left(|X|^4\right)}{\sigma_X^4} = \frac{\Gamma\left(\frac{1}{s}, 0\right) \cdot \Gamma\left(\frac{5}{s}, 0\right)}{\Gamma^2\left(\frac{3}{s}, 0\right)} \tag{4.45}$$

Equations 4.44 and 4.45 can be derived by using 4.29 and some arithmetic manipulations. Given the monotonic nature of these functions (or their reciprocals), they could be solved to the precision of the quantized values of the shape parameter s using a small table with the values at the edges of the interval where each quantized value is contained (usually centered). If more precision is desired, this same table could be used as the starting point for a secant root finding algorithm, for example.

Log-linear plots of the reciprocals of the square root of these two ratios are shown in Figure 4.21 for a wide range of values of the shape parameter s. It is important to note that there are many other ways of determining the shape parameter of an exponential power distribution but the two methods given above are quite straightforward and fast to calculate.

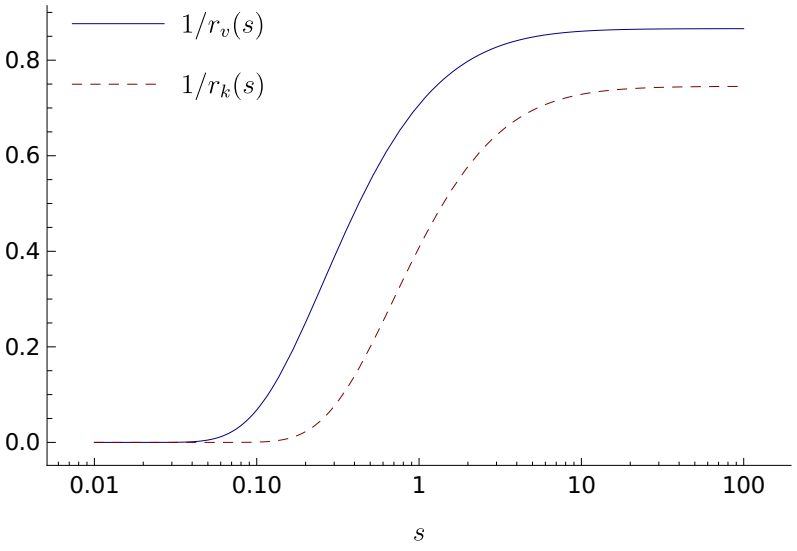


FIGURE 4.21: Reciprocal EPD $r_v(s)$ and $r_k(s)$

Theoretically, if the random variable X is actually distributed exactly according to an exponential power distribution with shape parameter s, then both 4.44 and 4.45 should yield the same value of s. In practice, the proximity of their values could be used as an

indication of how well the data is fitted by an exponential power distribution and both are quite straightforward to calculate from the coefficient data.

4.12 Examples

The AC coefficients resulting from a 6-level decomposition of the grey level 512×512 image "lena" (Figure 4.22) were used as the input coefficients and the resulting tables for their distortion reduction per significant bit and per refinement BIT ($\eta_n = 1$) are shown below for the CDF-9/7 DWT with $\langle\sqrt{2}, \sqrt{2}\rangle$ and $\langle 1, 2\rangle$ normalizations.

FIGURE 4.22: Lena (512×512)

Table 4.5 shows the values for the CDF-9/7 with the $\langle\sqrt{2}, \sqrt{2}\rangle$ normalization (DC gain of $\sqrt{2}$ and an AC gain of $\sqrt{2}$) in which case it is an almost energy preserving transform but also an expansive one, as can be seen from the 13 bitplanes needed to encode all of its transform coefficients, including the LL subband (DC coefficients), and also from the measured value of its standard deviation $\sigma = 32.213$. The shape parameter for the AC coefficients was calculated as $s = 0.16016$ using r_v (4.44) and as $s = 0.20428$ using r_k (4.45). The quantized values used to create the table were $\sigma = 32$ and $s = 0.15$.

TABLE 4.5: Lena AC coeffs (6 × CDF-9/7 $\langle\sqrt{2}, \sqrt{2}\rangle$, $s = 0.15$, $\sigma = 32$)

0.62631												
0.49332	1.2024											
0.49757	0.98230	2.2965										
0.49917	0.99359	1.9537	4.3622									
0.49973	0.99783	1.9832	3.8800	8.2397								
0.49991	0.99928	1.9943	3.9564	7.6918	15.479							
0.49997	0.99977	1.9981	3.9853	7.8876	15.215	28.929						
0.49999	0.99992	1.9994	3.9952	7.9621	15.711	30.015	53.823					
0.50000	0.99998	1.9998	3.9984	7.9876	15.902	31.262	59.014	99.765				
0.50000	0.99999	1.9999	3.9995	7.9960	15.968	31.750	62.123	115.56	184.38			
0.50000	1.0000	2.0000	3.9998	7.9987	15.990	31.918	63.360	123.24	225.17	339.99		
0.50000	1.0000	2.0000	3.9999	7.9996	15.997	31.974	63.791	126.37	243.99	436.13	625.96	
0.50000	1.0000	2.0000	4.0000	7.9999	15.999	31.992	63.933	127.47	251.85	481.81	838.70	1151.2
0	1	2	3	4	5	6	7	8	9	10	11	12
						bitplane						

It is quite clear that the values for the distortion reduction per significant bit for bitplanes 6 to 12 are smaller than all values of the distortion reduction per refinement BIT at the same bitplane while for bitplane 5 it is smaller than all refinement values except for 1 and for bitplanes 0 to 4 it is finally greater than all refinement values.

The total number of AC coefficients in each bitplane (base 2 logarithm histogram) is given in table 4.6 where the first count is the number of coefficients that are rounded or truncated to 0 and are not encoded.

TABLE 4.6: Lena AC coeffs per bitplane (6 × CDF-9/7 $\langle\sqrt{2}, \sqrt{2}\rangle$, $s = 0.15$, $\sigma = 32$)

69175	55118	65356	40591	16290	8097	4165	1953	858	304	134	34	5	0
-	0	1	2	3	4	5	6	7	8	9	10	11	12
						bitplane							

Table 4.7 shows the values for the CDF-9/7 with the $\langle 1, 2\rangle$ normalization (DC gain of 1 and AC gain of 2). The measured value of the standard deviation was $\sigma = 7.1592$ and the calculated shape parameter was $s = 0.42821$ and $s = 0.45170$ using r_v (4.44) and r_k (4.45), respectively. The quantized values used to create the table were $\sigma = 2^{2+13/16} \approx 7.025$ and $s = 0.45$.

This table shows that the refinement values increase with the refinement level but are always smaller than their respective significant values ate the same bitplane. However, this transform is not energy preserving so that each BIT contributes a different amount of distortion reduction depending on the subband it is in. This means that distortion reduction for coefficients in each subband should be scaled by this subband's energy

gain, resulting in multiple tables, one per subband, in order to produce a rate distortion optimized order for all coefficients.

TABLE 4.7: Lena AC coeffs ($6 \times$ CDF-9/7 $\langle 1,2 \rangle$, $s = 0.45$, $\sigma \approx 7.025$)

0.73344							
0.49373	1.3792						
0.49651	0.97378	2.5378					
0.49817	0.98572	1.8958	4.5454				
0.49906	0.99258	1.9431	3.6033	7.8984			
0.49953	0.99623	1.9704	3.7788	6.5612	13.331		
0.49976	0.99811	1.9850	3.8835	7.1649	11.110	22.004	
0.49988	0.99906	1.9925	3.9408	7.5506	12.987	16.779	35.905
0	1	2	3	4	5	6	7
bitplane							

The total number of AC coefficients in each bitplane (base 2 logarithm histogram) is given in table 4.8 where the first count is the number of coefficients that are rounded or truncated to 0 and are not encoded.

TABLE 4.8: Lena AC coeffs per bitplane ($6 \times$ CDF-9/7 $\langle 1,2 \rangle$, $s = 0.45$, $\sigma \approx 7.025$)

66477	52531	64913	47533	20397	7686	2296	244	3
-	0	1	2	3	4	5	6	7
bitplane								

4.13 Summary

General formulae for the distortion reduction per significant and refinement BITs and bits were derived for a generic PDF $p(x)$. In fact, 4.7 and 4.11 describe the distortion reduction per BIT for significant and refinement coefficients, respectively, and to the best of our knowledge have not been presented in this form in the available literature. The same can also be said about 4.18 and 4.20 which describe the entropy per BIT for significant and refinement coefficients, respectively, for a general PDF $p(x)$.

Specific formulae for the Exponential Power Distribution were derived and the conditions when the distortion reduction per refinement BIT and per refinement bit are higher than their corresponding distortion reduction per significant bit were shown, allowing for the optimal rate distortion ordering of the significant and refinement information.

Specific closed formulae were given for the uniform and laplace distributions based on the results for the EPD PDF by using their respective values of the shape parameter $s \to \infty$ and $s = 1$. It was shown that the distortion reduction per refinement BIT for both distributions are the same for each bitplane and, as the refinement entropy per BIT $\eta_n = 1$ in the uniform case, the same also applies to its distortion reduction per bit.

It was shown that depending on the shape parameter of the resulting EPD PDF, the refinement distortion reduction may be higher than the significant distortion reduction for the same bitplane, which results in a rate distortion optimized ordering and, therefore, better embedding for low bit rate encoding in bandwidth constrained environments, e.g., sonar and RF underwater communication channels.

Chapter 5

The Depth Embedded Block-Tree (DEBT) Algorithm

The DEBT algorithm has been designed as a set-partitioning algorithm in which coefficients are grouped into sets which are expected to be composed of similar magnitude coefficients. These sets are either "block" sets, i.e., sets of coefficients in the same subband which try to take advantage of the correlation among neighboring coefficients, and "tree" sets, i.e., sets of coefficients that try to exploit the correlation produced by the energy clustering property of the wavelet transform at a spatial location in multiple resolutions across the same subband type.

Like any set-partitioning algorithm, as long as all coefficients that form a block or a tree are below a certain threshold, i.e., are insignificant in relation to this threshold, we only need a single value to convey this information for all coefficients in this set. Once this set becomes significant, it is broken down into smaller sets which are then recursively tested for significance at some later time in order to find out the coefficients that are actually significant.

The DEBT algorithm introduces the concept of variable depth blocks and trees so that a tree can be partitioned either into lower depth subtrees or decomposed into a block and a higher depth subtree. This is necessary because, even when a tree tests positive for significance, we may not be interested in the coefficients that are actually significant unless they are on the current subband (tree's top subband), which is given by the rate distortion optimized scanning list.

In fact, given that most biorthogonal wavelets have different gains for each subband in each decomposition level, tree testing may not be the best solution to the problem of minimizing the output stream in relation to progressiveness because, even though it could lead to a smaller stream in the long run, we may be conveying significance information for coefficients that will only be scanned much later and, therefore, would be wasting valuable space with information that will not help reduce distortion at this point. Also, trees are most valuable when used in conjunction with an energy preserving DWT so that each bitplane will be scanned across all subbands, allowing for a depth-first scanning order in search for the next significant coefficient.

In fact, when using an energy preserving DWT with either an uniform or laplace PDF for modelling the coefficients' distribution, the "orthogonal" scanning order depicted in Figure 5.1 used with alternating significant and refinement passes is usually the optimal ordering for an embedded algorithm.

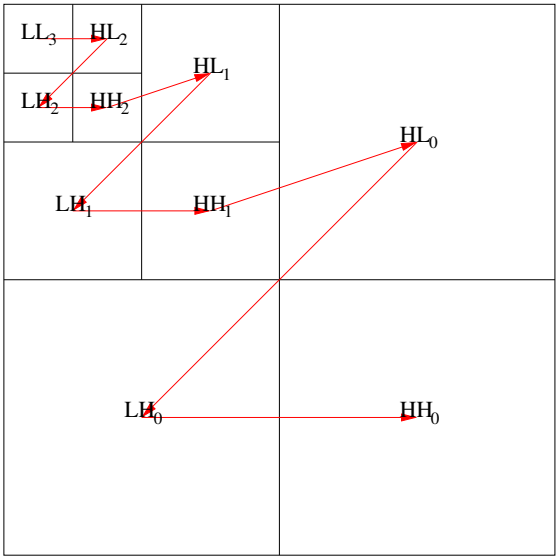


FIGURE 5.1: Orthogonal DWT band scanning order

This ordering is the one used by all existing set-partitioning algorithms, which are dependent on an energy preserving DWT for consistent results. However, none of the available well performing biorthogonal DWT are actually energy preserving with the CDF-9/7 being very close to one when used with a $\langle\sqrt{2}, \sqrt{2}\rangle$ normalization, in which case it is an expansive transform, i.e., has a gain greater than 1 for the LL subband, which increases the coefficients' dynamic range at each transformation level.

The orthogonal ordering shown in Figure 5.1 however, works very poorly when the DWT used is not energy preserving, which is usually the case for most DWT used for image compression, specially the ones used in the lossless case, i.e., the ones that map integers into integers. This means that coefficients at lower bitplanes on a lower resolution subband should be coded before ones at higher bitplanes on a higher resolution subband. A possible workaround is to scale each coefficient by its respective subband's gain, or a rational approximation to it for the lossless case, which invariably increases the transform's dynamic range.

The DEBT algorithm was designed to work with any biorthogonal DWT and, therefore, does not use this "orthogonal" scanning order. Also, the use of a PDF which is neither uniform nor laplacian does not allow it to use a single refinement pass for each bitplane on each subband or across subbands. As will be shown, coefficients are grouped by magnitude range (bitplane), subband, and either significance or refinement levels. Each group is assigned a value for its expected distortion decrease and inserted into a list that is finally sorted in decreasing distortion reduction order and used as the scanning order to output the information for each group.

The DEBT algorithm block diagram is shown in Figure 5.2 with its main blocks, which will be detailed in this chapter.

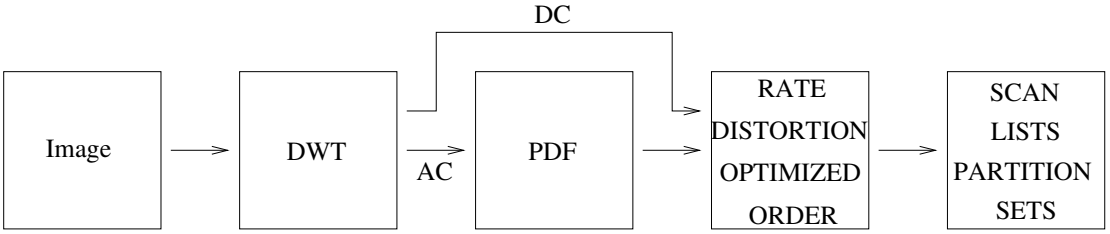


FIGURE 5.2: DEBT block diagram

Basically, the DEBT algorithm applies a DWT to the image coefficients, resulting in a coefficient matrix. The AC coefficients are modeled by an EPD PDF while the DC are assumed to be uniformly distributed. The rate distortion optimized order for the significant and refinement coefficients can then be calculated and is traversed in decreasing distortion order while partitioning the sets as necessary.

5.1 Representation of variable depth blocks and trees

A discrete image is defined as a rectangular array of pixels of width W and height H. The position of each pixel is represented by (i, j), where $0 \leq i < H$ (line) and $0 \leq j < W$ (column) with the origin $(0, 0)$ at the top left corner.

A transformation of this image will produce 4 subbands, $\{LL, HL, LH, HH\}$, as shown in Figure 5.3 where, by definition, LL is $\lceil \frac{W}{2} \rceil$ coefficients wide by $\lceil \frac{H}{2} \rceil$ coefficients high and HH is $\lfloor \frac{W}{2} \rfloor$ coefficients wide by $\lfloor \frac{H}{2} \rfloor$ coefficients high, where $\lceil . \rceil$ means the smallest integer greater than or equal to its argument (integer ceiling operation) and $\lfloor . \rfloor$ means the largest integer smaller than or equal to its argument (integer flooring operation). It is easy to see that HL has the same width as HH and the same height as LL and LH has the same width as LL and the same height as HH.

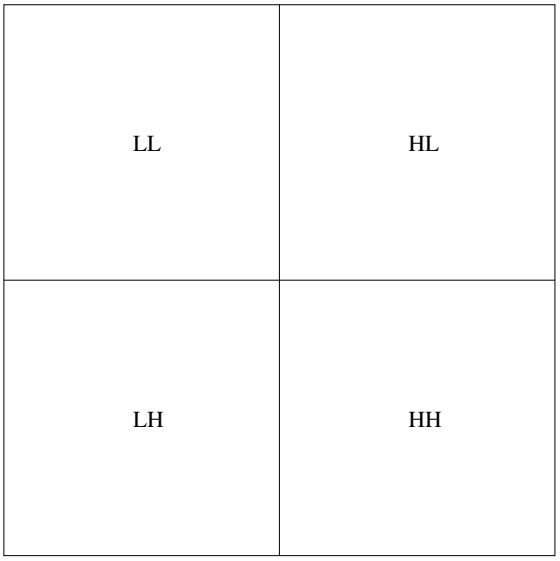


FIGURE 5.3: 1 level transformation

Each resulting LL subband from a transformation can be further transformed in a recursive way, generating four new subbands, until we reach a desired maximum number of transformations. The initial image, without any transformation applied, can be described as LL_0 and every transformation applied to an LL_n subband will generate 4 new subbands described as $\{LL_{n+1}, HL_n, LH_n, HH_n\}$, as seen on Figure 5.4, which shows a 3 level transformation applied to an image. After N transformations there are $3N + 1$ subbands left $\{LL_N, \{HL_n, LH_n, HH_n\} : 0 \leq n < N\}$.

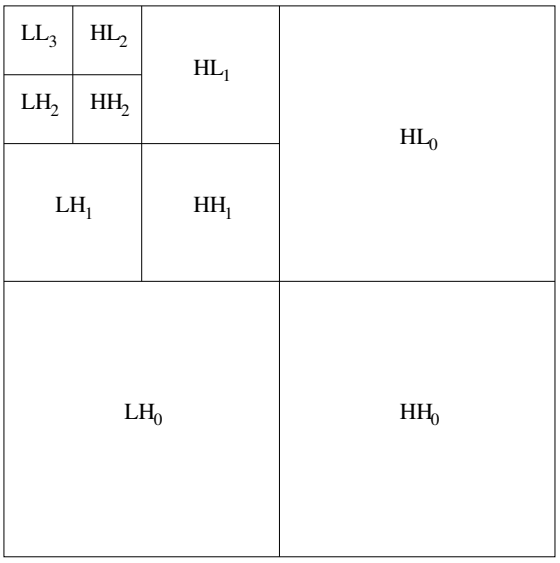


FIGURE 5.4: 3 level transformation

While, in principle, there is no limit to the number of transformations applied, it is clear that, for a dimension of length l, there will be only a single coefficient left after $\lceil \log_2(l) \rceil$ transformations are applied to each previous lowpass component of this dimension. By definition, a transformation applied to a dimension of length 1 is $\lceil \frac{1}{2} \rceil = 1$ and is, in fact, a null operation.

The maximum depth of a dimension is defined as being the maximum number of non null transformations that can be applied to this dimension, i.e.,

$$D(n) = \lceil \log_2(n) \rceil \tag{5.1}$$

is a property of the dimension itself and has no correlation to the number of transformations actually applied to an image dimension except for being a sensible upper bound for it.

This way, the maximum depth D for a $W \times H$ image is given by the maximum depth of both dimensions, i.e.,

$$D = \max(D(H), D(W)) = \max(\lceil \log_2(H) \rceil, \lceil \log_2(W) \rceil) \tag{5.2}$$

The maximum number of transformations M may reach up to the maximum depth but is usually defined as the minimum depth of both dimensions, i.e.,

$$M = \min(D(H), D(W)) = \min(\lceil \log_2(H) \rceil, \lceil \log_2(W) \rceil) \tag{5.3}$$

Figure 5.5 illustrates $D = 5$ for a 32×32 coefficient matrix. Once again, it is important to distinguish the maximum depth D from the actual number of transformations N applied to an image. The position $(\text{line} = i, \text{column} = j)$ of a set, together with its depth d, is always in relation to the maximum depth D, no matter the number of transformations actually applied (the actual number of transformations will define the depth of the initial blocks and trees).

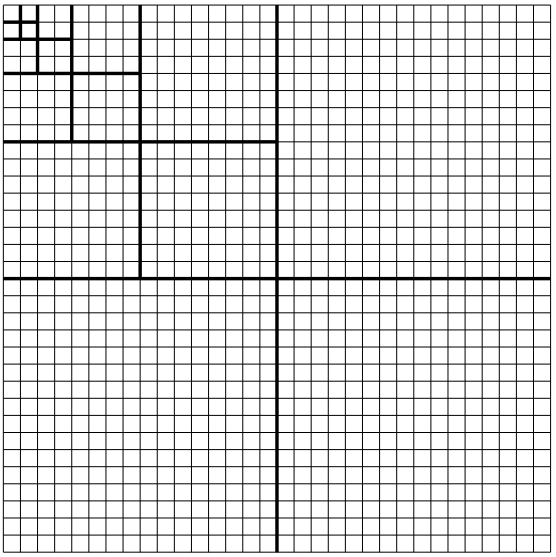


FIGURE 5.5: Maximum depth

Also, in order to simplify the block and tree definitions below, it is assumed that the image is a square array of pixels where the horizontal and vertical dimensions are the same power of 2 (the general case of arbitrary dimensions is explored in Appendix B).

We define a block $B^d_{(i,j)}$ of depth d at position (i, j) as the set of all coefficients $c_{(y,x)}$ such as:

$$B^d_{(i,j)} = \left\{ c_{(y,x)} : 2^d i \leq y < 2^d(i+1), 2^d j \leq x < 2^d(j+1) \right\} \tag{5.4}$$

It is important to note that the maximum depth d of a block is a function of its position (i, j) and the image dimensions, i.e., the maximum depth of $B^d_{(i,j)}$ is given by

$$\mathrm{dmax}(i, j) = D - \lceil \log_2(\max(i, j) + 1) \rceil \tag{5.5}$$

For example, in Figure 5.6,the dark gray 8×8 block at subband HL_0 could be described as $B^3_{(0,3)}$. For the middle gray blocks, the 16×16 block at subband HH_0, the 8×8 block at subband HH_1, and the 4×4 block at subband HH_2 can be described by $B^4_{(1,1)}$, $B^3_{(1,1)}$, and $B^2_{(1,1)}$ respectively. Finally, the light gray 8×8 block at subband LH_0 and the light gray 4×4 block at subband LH_1 can be described by $B^3_{(3,0)}$ and $B^2_{(3,0)}$ respectively.

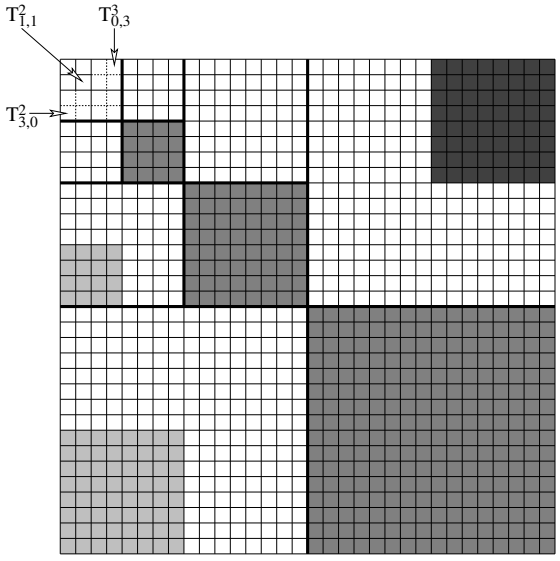


FIGURE 5.6: 3 level transformation sets

We define a tree $T^d_{(i,j)}$ of depth d at position (i, j) as the set of all blocks $B^z_{(i,j)}$ such as:

$$T^d_{(i,j)} = \left\{ B^z_{(i,j)} : d \leq z \leq \mathrm{dmax(i,j)} \right\} \tag{5.6}$$

The maximum depth of a tree is, just like a block, a function of its position (i, j) and the image dimensions and is also given by equation 5.5.

Trees are only used for the highpass subbands $\{\{HL_n, LH_n, HH_n\} : 0 \leq n < N\}$, i.e., the tree $T^d_{0,0}$ of depth d at position $(0, 0)$ would represent the set of all LL subbands starting from depth d which were replaced by successive transformations to previous LL

subbands. It is clear that after N transformations we are left with a single LL subband which is represented by an appropriate depth block $B^d_{(0,0)}$ at position $(0,0)$.

Figure 5.6 shows three trees $T^3_{(0,3)}$, $T^2_{(1,1)}$, and $T^2_{(3,0)}$ corresponding to the dark gray HL subband, middle gray HH subbands and light gray LH subbands, respectively. In fact, $T^3_{(0,3)} = B^3_{(0,3)}$, $T^2_{(1,1)} = \left\{B^2_{(1,1)}, B^3_{(1,1)}, B^4_{(1,1)}\right\}$, and $T^2_{(3,0)} = \left\{B^2_{(3,0)}, B^3_{(3,0)}\right\}$.

With the above definitions, we can describe any $W \times H$ image which has been transformed in N levels by the 4 element set

$$I = \left\{B^d_{(0,0)}, T^d_{(0,1)}, T^d_{(1,0)}, T^d_{(1,1)}\right\}, \quad d = \mathrm{D} - N \tag{5.7}$$

For example, if a 32×32 image is transformed in 3 levels then $d = 5 - 3 = 2$ and the whole image can be described by $\{B^2_{(0,0)}, T^2_{(0,1)}, T^2_{(1,0)}, T^2_{(1,1)}\}$.

The initial set is only dependent on the image dimensions and the number of transformations and both encoder and decoder, with the same knowledge of these values, can insert these sets into appropriate lists in the same manner so the initial state is the same for both. Then, a series of significance tests and set partitionings occur on each element of this list, allowing the decoder to exactly follow the encoder steps and reconstruct the original coefficient matrix to any desired precision.

It is important to note that $B^0_{(i,j)} = c_{(i,j)}$ (a depth 0 block is the coefficient at its position) and that $T^{\mathrm{dmax}(i,j)}_{(i,j)} = B^{\mathrm{dmax}(i,j)}_{(i,j)}$ (a tree composed of only one block is the same as the block).

Also, given a set $S^d_{(i,j)}$ (either a block or a tree) of depth d at position (i,j) of an N-level transformed image, we can find out the exact subband where its top level coefficients are by calculating

$$\begin{cases} b = \mathrm{dmax}(i,j) - d \\ v = \lceil \log_2(i+1) \rceil \\ h = \lceil \log_2(j+1) \rceil \end{cases} \tag{5.8}$$

and the subband where the set's top block coefficients are located is found by

$$\begin{cases} LL_N & , b \geq N \\ HL_b & , 0 \leq b < N, h > v \\ LH_b & , 0 \leq b < N, h < v \\ HH_b & , 0 \leq b < N, h = v \end{cases} \tag{5.9}$$

5.2 Set Partition and Decomposition

We define the value of a set S (either block or tree) $\|S\|$ as its L^∞ norm, also known as the supremum norm or simply infinity norm, $\|S\|_\infty = \max\{|c_{(i,j)}| : c_{(i,j)} \in S\}$. Therefore, the value of a block or tree is simply the maximum absolute value of all its coefficients.

A set S (block or tree) is said to be significant in relation to a threshold L if its value is greater than or equal to this threshold, i.e., $\|S\|_\infty \geq L$, otherwise if is said to be insignificant.

Whenever we find a significant set (block or tree), we must partition it into smaller sets until we reach the actual coefficients whose modulus are greater than or equal to this threshold and send their sign information. This is the actual information that will decrease the reconstruction error (along with the next bitplanes of all significant coefficients), since all other information is actually addressing information (significance map), necessary for the decoder to find out the position of the significant coefficients.

Even though there are many different ways a set can be partitioned and, in fact, a previous analysis phase would be able to tell which is the best way to do it in the current context, quadrature partitioning performs quite well.

A significant block can be partitioned into 4 subblocks, which can then be tested individually, until a depth 0 block (coefficient) is reached, i.e.,

$$B^d_{(i,j)} = \left\{ B^{d-1}_{(2i,2j)}, B^{d-1}_{(2i,2j+1)}, B^{d-1}_{(2i+1,2j)}, B^{d-1}_{(2i+1,2j+1)} \right\}, \quad 0 < d \leq \text{dmax(i,j)} \tag{5.10}$$

A tree, on the other hand, can be partitioned, just like a block, into 4 subtrees, which can then be tested individually, until a depth 0 tree (first block is a coefficient) is reached

$$T^d_{(i,j)} = \left\{ T^{d-1}_{(2i,2j)}, T^{d-1}_{(2i,2j+1)}, T^{d-1}_{(2i+1,2j)}, T^{d-1}_{(2i+1,2j+1)} \right\}, \quad 0 < d \leq \text{dmax}(i,j) \tag{5.11}$$

or can be decomposed into its top block and remaining tree, which can then be tested individually, until the tree is of depth $\text{dmax}(i,j)$ in which case it becomes indistinguishable from a block

$$\begin{cases} T^d_{(i,j)} = \left\{ B^d_{(i,j)}, T^{d+1}_{(i,j)} \right\} & , 0 \leq d < \text{dmax}(i,j) \\ T^d_{(i,j)} = B^d_{(i,j)} & , d = \text{dmax}(i,j) \end{cases} \tag{5.12}$$

It is easy to see that tree decomposition (5.12) follows from the tree definition (5.6) by removing the first block $B^d_{(i,j)}$ from the original tree $T^d_{(i,j)}$ and noting that the remaining set $\left\{ B^z_{(i,j)} : d+1 \leq z \leq \text{dmax}(i,j) \right\}$ is, by definition, $T^{d+1}_{(i,j)}$.

We will reserve the term "tree partition" for the division of a tree into its 4 subtrees (5.11) and "tree decomposition" for the division of a tree into its top block and remaining tree (5.12). Figures 5.7 and 5.8 illustrate, respectively, the partition and decomposition of a tree $T^2_{(i,j)}$ of depth 2.

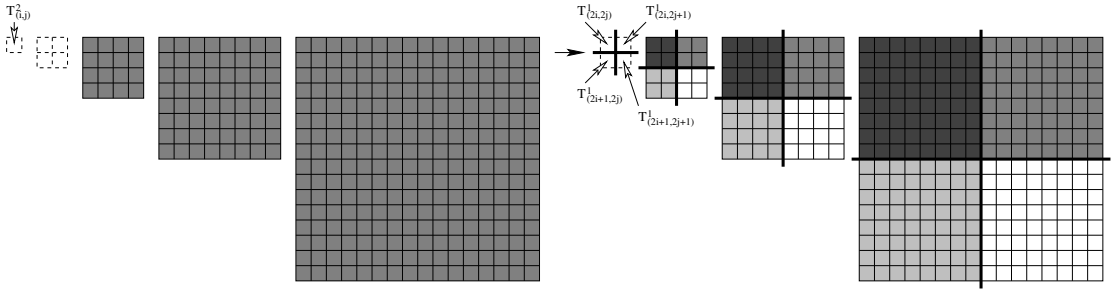


FIGURE 5.7: Tree partition

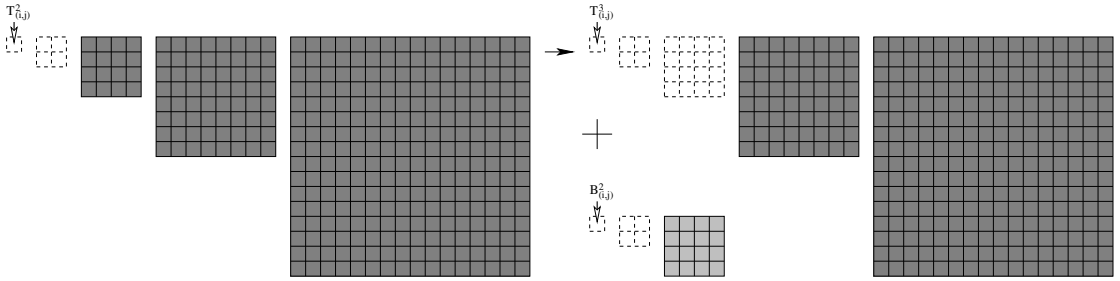


FIGURE 5.8: Tree decomposition

The choice between tree partition or decomposition is what allows us to explore the intra and inter band coefficient correlations in a dynamic way, better adapting to the characteristics of certain parts of the image and its spatial and spectral contents.

However, as sets are built based solely on the magnitude of the coefficients and do not take into consideration neither the coefficients' distribution nor the subband gains of the used DWT, tree sets are best used when the subband gains are unitary, as in the case of orthogonal DWT or the "near orthogonal" CDF-9/7 DWT [44] with a $\langle\sqrt{2}, \sqrt{2}\rangle$ normalization, when a depth-first search can be made to actually output the significant coefficient no matter the subband it is located at. In any case, if the probability of insignificant trees is high then it might actually be beneficial to code their insignificance.

5.3 Lists of Sets and Refinement Bits

Lists are used so that both encoder and decoder can keep the same state regarding set partitioning and decomposition along with the order in which the refinement information will be conveyed. This way, by following the decisions conveyed by the encoder regarding the significance of sets, the decoder can build the exact same lists as the encoder and synchronize their execution.

Even though lists are used for the description of the algorithm, the implementation may choose to used different data structures or even simpler queues where the only operation is to append new elements at the end with a fixed size memory pool that never needs resizing. Also, the amount and kind of lists described here are used in order to describe the algorithm and are not necessarily an implementation specification.

Defining $A = \left\| B^D_{(0,0)} \right\|_\infty$ as the maximum absolute value of the transformation matrix, the number of bitplanes B is given by

$$B = \lceil \log_2(A + 1) \rceil \tag{5.13}$$

and, together with the number of transformations N applied to the image, a matrix of cells is defined for each $\{LL, HL, LH, HH\}$ frequency component.

Figure 5.9 shows an example cell matrix for the case when $N = 6$ and $B = 9$. In this figure, only one 6×9 matrix is shown but there is one for each of the highpass subband

components $\{HL, LH, HH\}$. Each cell is composed of a list of sets (LS) and $B-1-\mathbf{bp}$ lists of refinement bits (LR_k), where **bp** is the bitplane number, which are traversed in the order given by the scanning list (the total number of lists is equal to the number of elements in the scanning list, i.e., $(3N+1)B(B+1)/2$).

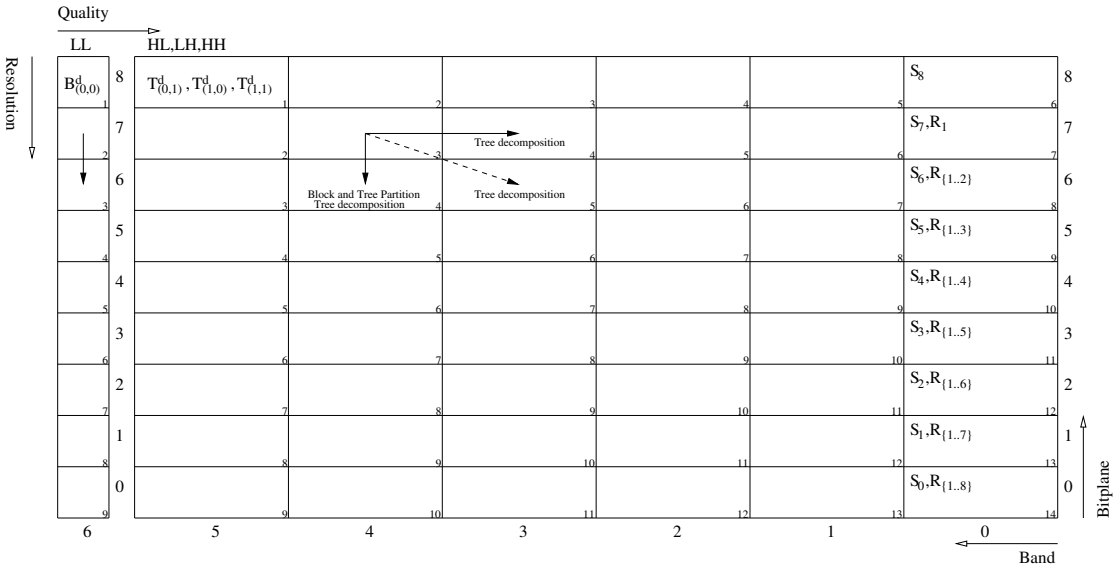


FIGURE 5.9: Set Scan Matrix

Each element of the scanning list is composed of the tuple $\langle \mathbf{bp}, \mathbf{bd}, \mathbf{sb}, \mathbf{sr}, \mathbf{rl} \rangle$ where **bp** is the bitplane ($0 \leq \text{bp} < B$), **bd** is the band ($0 \leq \text{bd} < N$), **sb** is the subband type (LL, HL, LH, HH), **sr** is a boolean value indicating if it is significance or refinement, and **rl** is the refinement level ($0 < \text{rl} < B$).

For each tuple of the scanning list, the appropriate cell in the set scan matrix is selected and either the significance or refinement list is traversed depending on the value of **sr**. When **sr** indicates that this is a significance scan, the list containing the sets to be tested is traversed, the testing results are appended to the output stream, the generated subsets from partition and decomposition are appended to the either the next band or next bitplane cell's list of sets, and the significant coefficients' refinement bits are appended to their respective lower bitplane cells' lists of refinement bits.

In order to facilitate the description of the algorithm, each cell in the matrix in Figure 5.9 is supposed to contain a list of sets LS and lists of refinement bits LR_k. It is important to note that, even though the description in here revolves around a large number of lists, a good implementation may set a maximum bound on the lists sizes and allocate it all at once, in the beginning of the algorithm. Then each cell will contain only

pointers to the start and end of each list in the previously allocated memory pool and the only operation needed is the appending of new elements to existing lists that require no expansion. Also, the many lists of refinement bits could be easily implemented using a single list of significant coefficients for each band.

5.4 List Traversal

Initially, each element of the set I (5.7), which covers the whole image, is inserted in its respective list of sets $LS(\text{band}, \text{bitplane})$ and the initial threshold is set to 2^{B-1}, where B is given by 5.13, as depicted in Figure 5.10. As previously defined, the LL subband index (band number), by definition, equals the number of transformations but is not required since there is only a single LL subband.

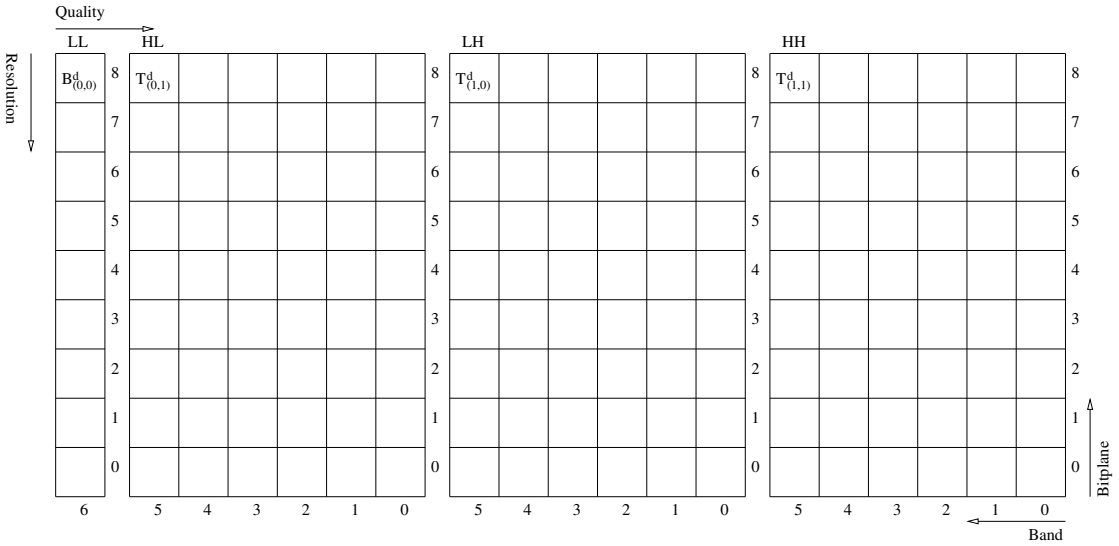


FIGURE 5.10: Matrix of Sets

Block partition can only generate subsets in the next bitplane (lower cell) because blocks are restricted to sets of coefficients in the same band. Trees, on the other hand, can be partitioned or decomposed and can generate subsets in the next band (right cell), in the next bitplane (lower cell), or even on the next band and bitplane (lower-right cell) as depicted in Figure 5.9. This means that in order for the list of sets LS of a cell to be traversed, it is necessary that both its left and upper cells' LS have already been traversed. The procedure that creates the scanning order list should take this into account and make sure this is the case. Also, any refinement list in a cell should only

be traversed strictly after its upper cell's list of sets has been traversed. In case only blocks and no trees are used, it is only necessary for the higher cell's LS to have been reversed in order to traverse the current cells' LS.

In order to increase parallelism and allow the execution of the previous top and left cells in any order and simultaneously, each list of sets LS could be divided into two lists: the list of top sets LTS and the list of left sets LLS. The LTS is used to keep the insignificant sets (blocks and trees) coming from the processing of the previous top cell (same band and previous bitplane) while the LLS is used to keep both significant and insignificant sets (trees) from the processing of the previous left cell (same bitplane and previous band). If the implementation is not a parallel implementation, it is sufficient to use a single list of sets LS, as long as the scanning order is the same as the one used in the parallel implementation and care is taken when allowing tree decomposition to generate sets on the next lower band and next lower bitplane (dashed line on Figure 5.9).

It is important to note that sets that belong to a column of this matrix have their top block in the band corresponding to the index of the column, i.e., they have their root at a decomposition level equal to its depth plus its column index (top block band). This means that block sets may change their bitplane (row) when partitioned but will always stay in the same band (column).

Even though it is possible to do depth-first searches on trees looking for the coefficients which are significant, like it is done in the SPIHT [25] algorithm, this would most likely violate the scanning order being used. In fact, the use of trees could be avoided entirely without any modifications to the algorithm by using the tree definition in 5.6 and inserting its respective blocks in the initial lists, as shown in Figure 5.11.

The use of only blocks has the advantage of never sending any information regarding other subbands when searching for significant coefficients in the current subband but also increases the number of initial sets which need to be tested, growing from 4 sets (1 block and 3 trees) to $3N + 1$ blocks. Also, the use of only block sets is simpler, faster, and allows for more parallelism. However, there may be occasions in which the use of trees is advantageous, specially when they are insignificant but, in general, there is little difference between using only blocks or using blocks and trees.

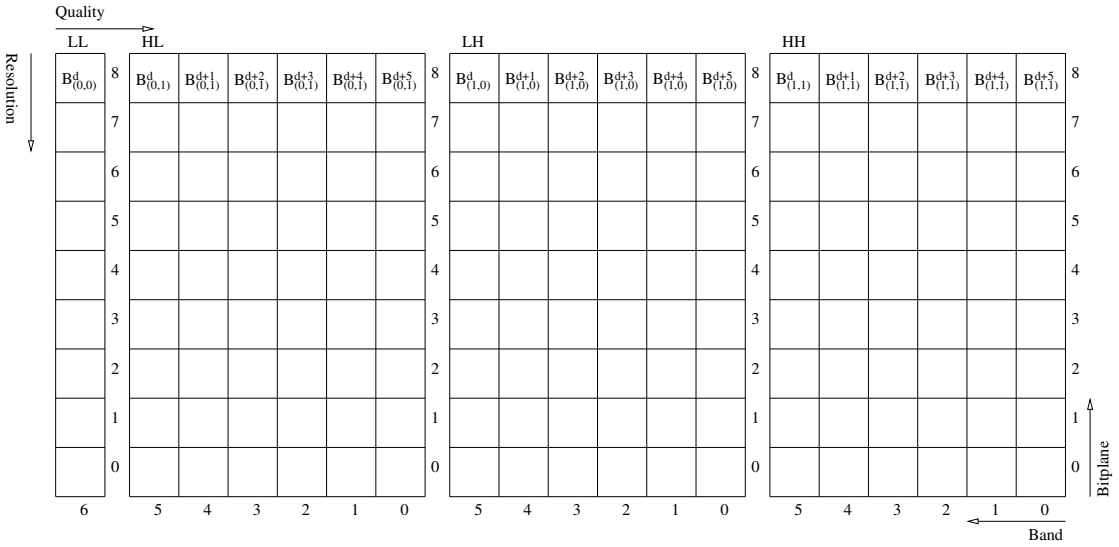


FIGURE 5.11: Matrix of Blocks

The results show that using only blocks or blocks and trees are almost equivalent and indicate that the inter band correlation is not being used effectively, which seems to be the case. When only blocks are used, each band is treated independently which means that any knowledge that could have been gained from previous or next bands are being ignored. It looks like some probabilistic modeling coupled with entropy coding is necessary to take advantage of the inter band dependencies but this would impact the performance of the algorithm both from the cost of using entropy coding coupled with a statistical model and also by the need of serialization (lack of parallelism) due to the dependencies between bands. Trees are most effectively used for depth-first searches which would only be an option for coefficients generated by an energy preserving DWT (equal gains for all subbands) so that the next coefficient to be encoded is the largest one, irrespective of the subband.

Set partition and decomposition allows for the encoding of many different orderings for the coefficients' significant and refinement bits and permits either a quality, resolution, or any other ordering which observes the cell ordering restrictions while maintaining a compact representation of the significance map. Even though the scanning order could be embedded in the stream itself, its cost is quite large, and a better solution is to use predefined scanning orders that both encoder and decoder can agree on.

5.5 Algorithm Outline

The DEBT algorithm can be broadly described by the following main steps:

1. Calculate the mean $\bar{m}$, subtract it from the image and send it as side information - if speed is of paramount importance, this step could be skipped by using a constant value for the mean, e.g., 128 for 8-bit images;

2. Transform the image with a suitable discrete wavelet transform $\bar{t}$ into $\bar{n}$ decompositions (bands) and send both $\bar{t}$ and $\bar{n}$ as side information;

3. Find the maximum absolute value A of the transform coefficients and send the number of bitplanes $\bar{b} = \lceil \log_2(A+1) \rceil$ as side information;

4. Fit an exponential power distribution to the high-pass wavelet coefficients (HL, LH, HH), quantize the values of the shape parameter $\bar{s}$ and standard deviation $\bar{\sigma}$ and send them as side information - if speed is of paramount importance use a predefined distribution (e.g. uniform or laplace). The low-pass coefficients (LL) usually use a predefined uniform distribution;

5. Assign values for the distortion decrease per bit (Table 4.1) for significance and refinement BITs for all coefficient ranges $\{[2^k, 2^{k+1}), 0 \leq k < \bar{b}\}$ resulting in the table $\mathcal{RDT}$. In general there are $\bar{b}(\bar{b}+1)/2$ weights which are a function of $\bar{b}$, $\bar{s}$, and $\bar{\sigma}$;

6. Assign values for the distortion decrease per bit for significance and refinement BITs for all subbands by multiplying table $\mathcal{RDT}$ by each subband gain for the appropriate DWT (Appendix A) resulting in the list $\mathcal{RDL}$ which, in general, has $(3\bar{n}+1)\bar{b}(\bar{b}+1)/2$ elements, and sort it in decreasing order (rate-distortion optimized order). The list $\mathcal{RDL}$ is a function of $\bar{t}$, $\bar{n}$, $\bar{b}$, $\bar{s}$, and $\bar{\sigma}$.

7. Set up the initial sets (state) composed of the LL block set $B^d_{0,0}$, and the HL, LH, and HH tree sets $T^d_{0,1}$, $T^d_{1,0}$, $T^d_{1,1}$, respectively, as shown in Figure 5.10; the initial sets' depth d is a function of the image dimensions $\bar{w}$ and $\bar{h}$ and the number of decompositions $\bar{n}$; If using only blocks, the initial trees are substituted by the blocks which define them (5.6) as shown in Figure 5.11.

8. Scan the $\mathcal{RDL}$ list and send the respective information for each element, i.e., traverse the list of sets or list of refinement bits depending on the value of the $\mathcal{RDL}$ element being processed. In the general case the $\mathcal{RDL}$ list contains $(3\bar{n}+1)\bar{b}(\bar{b}+1)/2$ elements in decreasing order of its weight $w_{s,r}$, $r <= s$ (Table 4.1) multiplied by the gain for each subband $\{\mathrm{LL}, \{\mathrm{HL}_k, \mathrm{LH}_k, \mathrm{HH}_k : 0 \leq k < \bar{n}\}\}$.

5.6 Set Partition and Decomposition Algorithm

This detailed procedures for each each element of the $\mathcal{RDL}$ list, which is composed of the tuple $\langle \mathbf{sb}, \mathbf{bp}, \mathbf{bd}, \mathbf{sr}, \mathbf{rl} \rangle$ where **sb** is the subband type, **bp** is the bitplane, **bd** is the band, **sr** is the significance or refinement selector, and **rl** is the refinement level, is given in this section and each set, block or tree, contains its position (y, x), its depth d, and a boolean value s indicating whether it is known to be significant.

Function "output_cell" (Algorithm 17) simply calls the appropriate function depending if this cell corresponds to a significant or refinement cell.

Algorithm 17 Output Cell

```
1  function output_cell(Int subband, Int bitplane, Int band, Bool sig, Int reflevel) :
       Int
2      if sig then
3          return output_cell_sig_coeffs(subband, bitplane, band)
4      else
5          return output_cell_ref_bits(subband, bitplane, band, reflevel)
```

In the case of refinement cells, function "output_cell_ref_bits" (Algorithm 18) will output the refinement coefficient bits of a (bitplane,band,reflevel) tuple. It must be noted that in order to output the refinement coefficients of a (bitplane,band,reflevel) tuple, it is necessary to have previously called "output_cell_sig_coeffs" on the (bitplane+reflevel,band) pair.

Algorithm 18 Output Refinement Bits

```
1  function output_cell_ref_bits(Int subband, Int bitplane, Int band, Int reflevel) :
       Int
2      Int count <- 0
3      for each bit b in cell[subband, bitplane, band].LR(reflevel) do
4          output b
5          count <- count + 1
6      return count
```

In the case of significant cells, function "output_cell_sig_coeffs" (Algorithm 19) will output the significant coefficient bits of a (bitplane,band) pair.

Algorithm 19 Output Significant Coefficients

```
1  function output_cell_sig_coeffs(Int subband,Int bitplane,Int band) : Int
2      Int count <- 0
3      cell C <- cell[subband,bitplane,band]
4      for each set S in C.LTS + C.LLS do
5          count <- count + output_set(S,subband,bitplane,band)
6      return count
```

It must be noted that in order to output the significant coefficients of a (bitplane,band) pair, it is necessary to have previously called "output_cell_sig_coeffs" on the previous bitplane (bitplane+1,band) pair and, if trees are being used, also on the (bitplane,band+1) pair. It traverses the LS list and calls "output_set" for each set in the list.

Function "output_set" (Algorithm 20) checks if the set is known to be significant and, in case it is not, tests it and outputs the result. In case it is significant, it simply calls either "output_block" or "output_tree" depending on the type of the set.

Algorithm 20 Output Set

```
1  function output_set(Set S,Int subband,Int bitplane,Int band) : Int
2      if not S.sig then
3          if ||S|| < 2^bitplane then
4              output 0
5              if bitplane > 0 then
6                  insert S in cell[subband,bitplane-1,band].LTS
7              return 0
8          else
9              output 1
10             S.sig <- 1
11     if S is BLOCK
12         return output_block(S,subband,bitplane,band)
13     else
14         return output_tree(S,subband,bitplane,band)
```

Function "output_tree" (Algorithm 21) uses an auxiliary function "partition" which returns true if this tree is to be partitioned or false if it is to be decomposed; it can either use static rules, in which case no information needs to be sent to the decoder, or dynamic rules, based on the data at hand only available to the encoder, in which case the decision should be relayed to the decoder.

If decomposing, a test is made to check if this tree is known to be significant and is composed of a single block (band = 0) and its top block is output (at this point, if the block is not significant, the call to "output_block" could be changed to a call to "output_set" in which case the block would be tested before being partitioned. The

Algorithm 21 Output Tree

```
1  # Returns the number of coeffs that became significant at this band-bitplane
2  function output_tree(Tree T,Int subband,Int bitplane,Int band) : Int
3      Int count <- 0
4      if partition(T,subband,bitplane,band) then
5          count <- output_block_tree(T,subband,bitplane,band)
6      else
7          Bool bsig <- T.sig and (band = 0)
8          B <- new Block(T.x,T.y,T.depth,bsig)
9          count <- output_block(B,subband,bitplane,band)
10         if band > 0 then
11             Bool stsig = T.sig and (count = 0)
12             ST = new Tree(T.x,T.y,T.depth+1,stsig)
13             insert ST in cell[subband,bitplane,band-1].LLS
14     return count
```

remaining subtree, which may be known to be significant by checking if the tree was significant and the top block was not, is then appended to the list of sets of the next band (right cell).

Function "output_block_tree" (Algorithm 22) partitions a set, either a block or a tree, into its immediate lower depth sets. It does not test the set in order to determine its significance but simply partitions it and, just like the "output_block_coeffs" function, takes advantage if the set is known to be significant by not sending the significance info in case only the last element of a significant set is in fact significant.

Algorithm 22 Output Block Tree

```
1  # Returns the number of coeffs that became significant at this band-bitplane
2  function output_block_tree(Set S,Int subband,Int bitplane,Int band) : Int
3    Int count <- 0
4    Int setcount <- 0
5    Rect R <- bounds(S.x,S.y,1,band)
6    for each Point p in R do
7      BT <- new Set(S.type,p.x,p.y,B.depth-1,false)
8      if ||BT|| < 2^bitplane then
9        output 0
10       if bitplane > 0 then
11         insert BT in cell[subband,bitplane-1,band].LTS
12     else
13       if (p is not the last point in R) or (not S.sig) or (setcount > 0) then
14         output 1
15       BT.sig <- true
16       count <- count + output_set(BT,subband,bitplane,band)
17       setcount <- setcount + 1
18   return count
```

Function "output_block" (Algorithm 23) simply test the block's depth d and calls either "output_block_coeffs" in case d is smaller than or equal to 1, otherwise if proceeds to partition the block by calling "output_block_tree".

Finally, function "output_block_coeffs" (Algorithm 24) outputs the significance (together with the sign in case it is significant) of every coefficient of a block. It is careful to check

Algorithm 23 Output Block

```
1  # Returns the number of coeffs that became significant at this band-bitplane
2  function output_block(Block B, Int subband, Int bitplane, Int band) : Int
3      Int count <- 0
4      if B.depth <= 1 then
5          count <- output_block_coeffs(B, subband, bitplane, band)
6      else
7          count <- output_block_tree(B, subband, bitplane, band)
8      return count
```

if this block is known to be significant and if all coefficients but the last one are not significant and, in this case, only outputs the sign because the test for significance of the last coefficient, in this case, is known to be true. The auxiliary function "bounds" returns a rectangle with the bounding box of a position (x, y) of depth d with the coefficients at band b.

Algorithm 24 Output Block Coefficients

```
1  # Returns the number of coeffs that became significant at this band-bitplane
2  function output_block_coeffs(Block B, Int subband, Int bitplane, Int band) : Int
3      Int count <- 0
4      Rect R <- bounds(B.x, B.y, B.depth, band)
5      for each Point p in R do
6          Int c <- coefficient(p.x, p.y)
7          if |c| < 2^bitplane then
8              output 0
9              if bitplane > 0 then
10                 insert new Block(p.x, p.y, 0, false) in cell[subband, bitplane-1, band
       ].LTS
11         else
12             if (p is not the last point in R) or (not B.sig) or (count > 0) then
13                 output 1
14             output sign(c)
15             count <- count + 1
16             if bitplane > 0 then
17                 insert c in cell[subband, bitplane-1, band].LR
18     return count
```

5.7 Examples

A 16×16 thumbnail version of the lena image shown in Figure 5.12 was obtained using the `netpbm` utility `pamscale` using the "`-width 16 -height 16`" parameters and was used as the input image for a run of the DEBT algorithm.

The pixel values corresponding to this image are shown in Table 5.1 which has its mean value of 128 subtracted from each pixel and then undergoes a 3 level ibior13x7 DWT transformation, resulting in the deinterleaved values shown in Table 5.2.

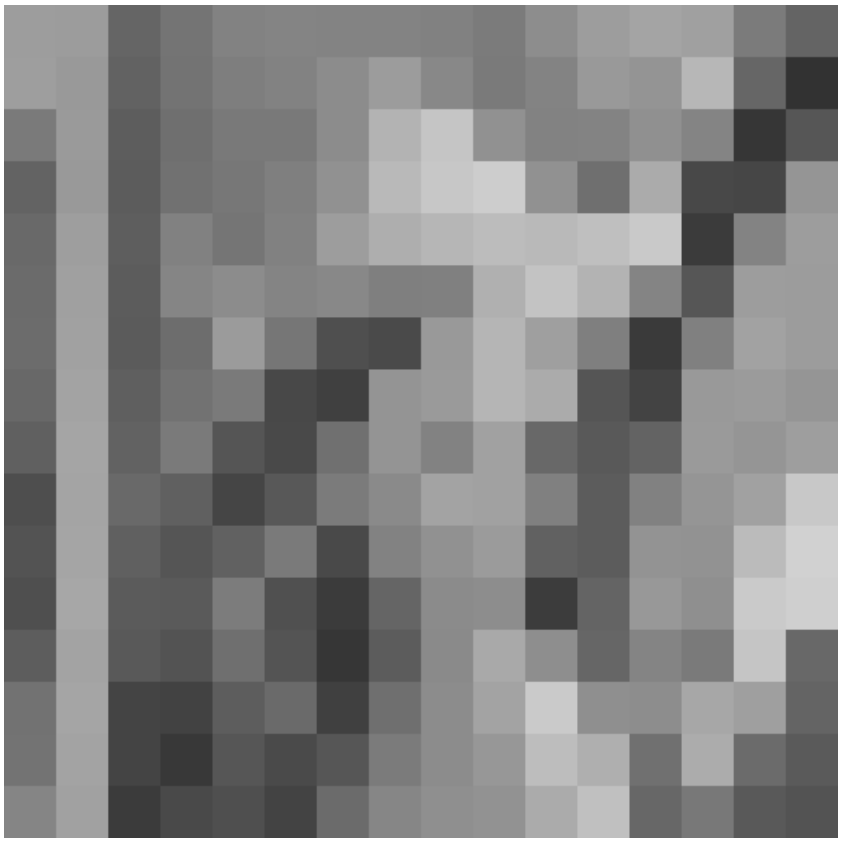


FIGURE 5.12: 16×16 Lena

The DEBT algorithm was used with only blocks (no trees), a fixed uniform PDF for the LL subband, and a fixed laplace PDF for the $\{HL_k, LH_k, HH_k : 0 \leq k < 3\}$ subbands in order to decrease the amount of information and simplify Table 5.3. Data is encoded according to the lines of this table (prioritization list) and a "0" is encoded each time a "$-$" or a "p" (positive sign) is encountered while a "1" is encoded each time a "$+$" or a "n" (negative sign) is encountered on the last column. The last column also lists the block $B^d_{y,x}$ at position (line $= y$, column $= x$) with depth d or the coefficient $c_{y,x}$ at position (line $= y$, column $= x$) which were tested preceding the actual output bit. Any time a block is tested insignificant is is inserted in its next lower bitplane list, otherwise it is partitioned and recursively tested.

The first column lists the set being encoded in the form $\mathbf{sb}^{\mathbf{bp}}_{\mathbf{bd}}$ where **sb** is the subband, **bd** is the band (subscript), and **bp** is the bitplane (superscript). The second column describes if this is a significant or refinement pass which, in the case of uniform and laplace distributions, includes all refinement levels for each bitplane.

TABLE 5.1: 16×16 Lena pixel values (mean = 128)

157	156	101	116	130	132	131	131	129	123	141	157	164	160	123	100
158	153	98	115	126	130	140	156	136	122	131	153	148	183	102	50
122	154	93	111	121	121	140	179	197	145	130	131	144	132	54	86
99	153	92	113	119	127	145	185	199	205	145	111	171	72	70	149
105	158	94	129	117	129	157	174	182	188	185	191	201	59	131	157
107	160	92	133	140	132	136	127	128	176	195	179	132	86	157	156
108	161	91	109	155	118	79	74	153	181	159	127	58	128	162	156
104	163	95	114	122	72	64	148	154	181	171	85	67	153	155	149
96	165	98	122	85	73	112	148	130	161	104	89	99	154	149	158
78	164	105	96	69	88	123	138	163	161	128	92	129	149	161	200
83	165	96	85	97	122	73	130	145	155	97	92	147	146	187	209
79	167	91	90	124	80	59	101	139	141	60	100	152	143	202	207
93	163	89	83	111	84	54	92	138	169	142	102	132	122	197	104
114	165	68	66	93	106	64	111	140	163	202	143	141	167	159	100
115	163	68	56	86	74	86	123	140	151	189	175	112	172	107	90
133	161	59	73	79	67	107	134	143	146	171	192	103	120	89	83

TABLE 5.2: 16×16 Lena ibior13x7 DWT values

-2	22	-23	-31	-41	6	-43	-68	19	7	-1	9	-9	11	30	-52
-22	2	-48	-20	-23	-24	20	-22	45	5	-7	8	-8	-15	37	43
-10	34	9	-42	8	-33	-5	91	61	28	-9	6	17	-17	-112	47
-9	3	1	33	-25	-49	-2	23	64	-15	-1	-15	7	15	26	-25
-5	-15	22	-44	8	-35	-61	-4	67	30	-27	28	28	-25	32	5
-11	13	-36	-12	-60	-82	71	90	79	-15	26	4	30	-17	-24	34
-9	9	-5	13	1	3	-51	-3	77	-20	6	2	22	-29	-37	-87
11	-20	32	1	-11	73	48	-65	68	-9	-3	14	-24	20	55	-16
9	-1	1	8	-18	3	9	4	-10	4	-2	13	6	13	26	-55
-8	-3	2	-8	17	-13	-24	-3	5	-10	6	2	41	-41	-5	46
0	4	1	22	-33	18	1	1	1	15	-9	20	2	2	-4	-10
1	-2	-4	-15	25	20	-8	2	0	-5	-3	53	-24	-42	9	-5
-10	6	-20	14	9	24	-1	-1	0	-2	-12	-38	-27	-14	-1	16
-1	0	7	-13	-2	-42	17	15	10	-2	-37	-7	12	34	3	45
10	-8	4	9	-1	21	16	11	2	6	36	10	-17	-26	10	5
8	-1	1	11	4	-13	-15	-21	-8	28	-13	-6	10	24	-53	1

Interestingly, even such a small image achieves lossless compression, resulting in a compressed size of 236 bytes, including header, markers, trailer, and padding used by our implementation. The header accounted for 12 bytes, the data itself consisted of 1771 bits which, after padding, resulted in 222 bytes, and a 2 byte trailer was appended to the stream resulting in a total length of 236 bytes.

A simple inspection of Table 5.3 reveals that the first 2 bitplanes (bitplanes 6 and 5) of the LL subband are the first ones to be tested, respectively, followed by bitplane 6 of

the coarser HL and LH subbands (which have the same weight) followed by the first refinement level (bitplane 5) of the LL subband. The ordering clearly shows the weight of the DWT subband gains (Table A.4) and the different gains of the uniform (Table 4.3) and laplace (Table 4.4) PDF used in the resulting ordering of the sets (lines).

TABLE 5.3: 16×16 Lena coding

set	**sr**	
LL_3^6	s	$B_{0,0}^1-$
LL_3^5	s	$B_{0,0}^1-$
HL_2^6	s	$B_{0,1}^1-$
LH_2^6	s	$B_{1,0}^1-$
LL_3^5	r	
LL_3^4	s	$B_{0,0}^1+\ c_{0,0}-\ c_{0,1}+p\ c_{1,0}+n\ c_{1,1}-$
HL_2^5	s	$B_{0,1}^1+\ c_{0,2}-\ c_{0,3}-\ c_{1,2}+n\ c_{1,3}-$
LH_2^5	s	$B_{1,0}^1+\ c_{2,0}-\ c_{2,1}+p\ c_{3,0}-\ c_{3,1}-$
HH_2^6	s	$B_{1,1}^1-$
HL_1^6	s	$B_{0,1}^2+\ B_{0,2}^1-\ B_{0,3}^1+\ c_{0,6}-\ c_{0,7}+n\ c_{1,6}-\ c_{1,7}-\ B_{1,2}^1-\ B_{1,3}^1+\ c_{2,6}-\ c_{2,7}+p$ $c_{3,6}-\ c_{3,7}-$
LH_1^6	s	$B_{1,0}^2-$
LL_3^4	r	
HL_2^5	r	
LH_2^5	r	
LL_3^3	s	$B_{0,0}^0-\ B_{1,1}^0-$
HL_2^4	s	$B_{0,2}^0+\ c_{0,2}n\ B_{0,3}^0+\ c_{0,3}n\ B_{1,3}^0+\ c_{1,3}n$
LH_2^4	s	$B_{2,0}^0-\ B_{3,0}^0-\ B_{3,1}^0-$
HL_0^6	s	$B_{0,1}^3+\ B_{0,2}^2+\ B_{0,4}^1-\ B_{0,5}^1-\ B_{1,4}^1+\ c_{2,8}-\ c_{2,9}-\ c_{3,8}+p\ c_{3,9}-\ B_{1,5}^1-\ B_{0,3}^2+$ $B_{0,6}^1-\ B_{0,7}^1-\ B_{1,6}^1-\ B_{1,7}^1\ c_{2,14}+n\ c_{2,15}-\ c_{3,14}-\ c_{3,15}-\ B_{1,2}^2+\ B_{2,4}^1+\ c_{4,8}+p$ $c_{4,9}-\ c_{5,8}+p\ c_{5,9}-\ B_{2,5}^1-\ B_{3,4}^1+\ c_{6,8}+p\ c_{6,9}-\ c_{7,8}+p\ c_{7,9}-\ B_{3,5}^1-\ B_{1,3}^2+$ $B_{2,6}^1-\ B_{2,7}^1-\ B_{3,6}^1-\ B_{3,7}^1\ c_{6,14}-\ c_{6,15}+n\ c_{7,14}-\ c_{7,15}-$
LH_0^6	s	$B_{1,0}^3-$
HH_2^5	s	$B_{1,1}^1+\ c_{2,2}-\ c_{2,3}+n\ c_{3,2}-\ c_{3,3}+p$
HL_1^5	s	$B_{0,2}^1+\ c_{0,4}+n\ c_{0,5}-\ c_{1,4}-\ c_{1,5}-\ B_{0,6}^0+\ c_{0,6}n\ B_{1,6}^0-\ B_{1,7}^0-\ B_{1,2}^1+\ c_{2,4}-$ $c_{2,5}+n\ c_{3,4}-\ c_{3,5}+n\ B_{2,6}^0-\ B_{3,6}^0-\ B_{3,7}^0-$

continued on next page

Table 5.3 – continued from previous page

set	**sr**	
LH_1^5	s	$B_{1,0}^2+$ $B_{2,0}^1-$ $B_{2,1}^1+$ $c_{4,2}-$ $c_{4,3}+n$ $c_{5,2}+n$ $c_{5,3}-$ $B_{3,0}^1-$ $B_{3,1}^1+$ $c_{6,2}-$ $c_{6,3}-$ $c_{7,2}+p$ $c_{7,3}-$
HH_1^6	s	$B_{1,1}^2+$ $B_{2,2}^1+$ $c_{4,4}-$ $c_{4,5}-$ $c_{5,4}-$ $c_{5,5}n$ $B_{2,3}^1+$ $c_{4,6}-$ $c_{4,7}-$ $c_{5,6}+p$ $c_{5,7}+p$ $B_{3,2}^1+$ $c_{6,4}-$ $c_{6,5}-$ $c_{7,4}-$ $c_{7,5}p$ $B_{3,3}^1+$ $c_{6,6}-$ $c_{6,7}-$ $c_{7,6}-$ $c_{7,7}n$
HH_2^5	r	
HH_0^6	s	$B_{1,1}^3-$
HL_2^4	r	$c_{1,2}+$
LH_2^4	r	$c_{2,1}-$
HL_1^5	r	$c_{0,7}-$ $c_{2,7}-$
LH_1^5	r	
LL_3^3	r	$c_{0,1}-$ $c_{1,0}-$
HL_0^5	s	$B_{0,4}^1+$ $c_{0,8}-$ $c_{0,9}-$ $c_{1,8}+p$ $c_{1,9}-$ $B_{0,5}^1-$ $B_{2,8}^0+$ $c_{2,8}p$ $B_{2,9}^0-$ $B_{3,9}^0-$ $B_{1,5}^1-$ $B_{0,6}^1-$ $B_{0,7}^1+$ $c_{0,14}-$ $c_{0,15}+n$ $c_{1,14}+p$ $c_{1,15}+p$ $B_{1,6}^1-$ $B_{2,15}^0+$ $c_{2,15}p$ $B_{3,14}^0-$ $B_{3,15}^0-$ $B_{4,9}^0-$ $B_{5,9}^0-$ $B_{2,5}^1-$ $B_{6,9}^0-$ $B_{7,9}^0-$ $B_{3,5}^1-$ $B_{2,6}^1-$ $B_{2,7}^1+$ $c_{4,14}+p$ $c_{4,15}-$ $c_{5,14}-$ $c_{5,15}+p$ $B_{3,6}^1-$ $B_{6,14}^0+$ $c_{6,14}n$ $B_{7,14}^0+$ $c_{7,14}p$ $B_{7,15}^0-$
LH_0^5	s	$B_{1,0}^3+$ $B_{2,0}^2-$ $B_{2,1}^2+$ $B_{4,2}^1-$ $B_{4,3}^1-$ $B_{5,2}^1+$ $c_{10,4}+n$ $c_{10,5}-$ $c_{11,4}-$ $c_{11,5}-$ $B_{5,3}^1-$ $B_{3,0}^2-$ $B_{3,1}^2+$ $B_{6,2}^1+$ $c_{12,4}-$ $c_{12,5}-$ $c_{13,4}-$ $c_{13,5}n$ $B_{6,3}^1-$ $B_{7,2}^1-$ $B_{7,3}^1-$
HH_2^4	s	$B_{2,2}^0-$ $B_{3,2}^0-$
HL_2^3	s	
LH_2^3	s	$B_{2,0}^0+$ $c_{2,0}n$ $B_{3,0}^0+$ $c_{3,0}n$ $B_{3,1}^0-$
LL_3^2	s	$B_{0,0}^0-$ $B_{1,1}^0-$
HH_1^5	s	$B_{4,4}^0-$ $B_{4,5}^0+$ $c_{4,5}n$ $B_{5,4}^0+$ $c_{5,4}n$ $B_{4,6}^0+$ $c_{4,6}n$ $B_{4,7}^0-$ $B_{6,4}^0-$ $B_{6,5}^0-$ $B_{7,4}^0-$ $B_{6,6}^0+$ $c_{6,6}n$ $B_{6,7}^0-$ $B_{7,6}^0+$ $c_{7,6}p$
HL_1^4	s	$B_{0,5}^0-$ $B_{1,4}^0+$ $c_{1,4}n$ $B_{1,5}^0+$ $c_{1,5}n$ $B_{1,6}^0+$ $c_{1,6}p$ $B_{1,7}^0+$ $c_{1,7}n$ $B_{2,4}^0-$ $B_{3,4}^0+$ $c_{3,4}n$ $B_{2,6}^0-$ $B_{3,6}^0-$ $B_{3,7}^0+$ $c_{3,7}p$
LH_1^4	s	$B_{2,0}^1-$ $B_{4,2}^0+$ $c_{4,2}p$ $B_{5,3}^0-$ $B_{3,0}^1+$ $c_{6,0}-$ $c_{6,1}-$ $c_{7,0}-$ $c_{7,1}n$ $B_{6,2}^0-$ $B_{6,3}^0-$ $B_{7,3}^0-$
HL_0^5	r	$c_{3,8}-$ $c_{2,14}+$ $c_{4,8}-$ $c_{5,8}-$ $c_{6,8}-$ $c_{7,8}-$ $c_{6,15}-$
LH_0^5	r	

continued on next page

Table 5.3 – continued from previous page

set	sr	
HH_0^5	s	$B_{1,1}^3+\ B_{2,2}^2+\ B_{4,4}^1-\ B_{4,5}^1-\ B_{5,4}^1-\ B_{5,5}^1\ c_{10,10}-\ c_{10,11}-\ c_{11,10}-\ c_{11,11}p$ $B_{2,3}^2+\ B_{4,6}^1+\ c_{8,12}-\ c_{8,13}-\ c_{9,12}+p\ c_{9,13}+n\ B_{4,7}^1+\ c_{8,14}-\ c_{8,15}+n\ c_{9,14}-$ $c_{9,15}+p\ B_{5,6}^1+\ c_{10,12}-\ c_{10,13}-\ c_{11,12}-\ c_{11,13}n\ B_{5,7}^1-\ B_{3,2}^2+\ B_{6,4}^1-\ B_{6,5}^1+$ $c_{12,10}-\ c_{12,11}+n\ c_{13,10}+n\ c_{13,11}-\ B_{7,4}^1-\ B_{7,5}^1+\ c_{14,10}+p\ c_{14,11}-\ c_{15,10}-$ $c_{15,11}-\ B_{3,3}^2+\ B_{6,6}^1+\ c_{12,12}-\ c_{12,13}-\ c_{13,12}-\ c_{13,13}p\ B_{6,7}^1+\ c_{12,14}-\ c_{12,15}-$ $c_{13,14}-\ c_{13,15}p\ B_{7,6}^1-\ B_{7,7}^1+\ c_{14,14}-\ c_{14,15}-\ c_{15,14}+n\ c_{15,15}-$
HH_2^4	r	$c_{2,3}-\ c_{3,3}-$
HH_1^5	r	$c_{5,5}-\ c_{5,6}-\ c_{5,7}-\ c_{7,5}-\ c_{7,7}-$
HL_1^4	r	$c_{0,7}-\ c_{2,7}+\ c_{0,4}-\ c_{0,6}-\ c_{2,5}-\ c_{3,5}+$
LH_1^4	r	$c_{4,3}-\ c_{5,2}-\ c_{7,2}-$
HL_2^3	r	$c_{1,2}-\ c_{0,2}-\ c_{0,3}+\ c_{1,3}-$
LH_2^3	r	$c_{2,1}-$
HL_0^4	s	$B_{0,8}^0+\ c_{0,8}p\ B_{0,9}^0-\ B_{1,9}^0-\ B_{0,5}^1-\ B_{2,9}^0+\ c_{2,9}p\ B_{3,9}^0-\ B_{1,5}^1-\ B_{0,6}^1-\ B_{0,14}^0+$ $c_{0,14}p\ B_{1,6}^1+\ c_{2,12}+p\ c_{2,13}+n\ c_{3,12}-\ c_{3,13}-\ B_{3,14}^0+\ c_{3,14}p\ B_{3,15}^0+\ c_{3,15}n$ $B_{4,9}^0+\ c_{4,9}p\ B_{5,9}^0-\ B_{2,5}^1+\ c_{4,10}+n\ c_{4,11}+p\ c_{5,10}+p\ c_{5,11}-\ B_{6,9}^0+\ c_{6,9}n$ $B_{7,9}^0-\ B_{3,5}^1-\ B_{2,6}^1+\ c_{4,12}+p\ c_{4,13}+n\ c_{5,12}+p\ c_{5,13}+n\ B_{4,15}^0-\ B_{5,14}^0+\ c_{5,14}n$ $B_{3,6}^1+\ c_{6,12}+p\ c_{6,13}+n\ c_{7,12}+n\ c_{7,13}+p\ B_{7,15}^0+\ c_{7,15}n$
LH_0^4	s	$B_{2,0}^2+\ B_{4,0}^1-\ B_{4,1}^1-\ B_{5,0}^1-\ B_{5,1}^1\ c_{10,2}-\ c_{10,3}+p\ c_{11,2}-\ c_{11,3}-\ B_{4,2}^1+\ c_{8,4}+n$ $c_{8,5}-\ c_{9,4}+p\ c_{9,5}-\ B_{4,3}^1+\ c_{8,6}-\ c_{8,7}-\ c_{9,6}+n\ c_{9,7}-\ B_{10,5}^0+\ c_{10,5}p\ B_{11,4}^0+$ $c_{11,4}p\ B_{11,5}^0+\ c_{11,5}p\ B_{5,3}^1-\ B_{3,0}^2+\ B_{6,0}^1-\ B_{6,1}^1+\ c_{12,2}+n\ c_{12,3}-\ c_{13,2}-$ $c_{13,3}-\ B_{7,0}^1-\ B_{7,1}^1-\ B_{12,4}^0-\ B_{12,5}^0+\ c_{12,5}p\ B_{13,4}^0-\ B_{6,3}^1+\ c_{12,6}-\ c_{12,7}-$ $c_{13,6}+p\ c_{13,7}-\ B_{7,2}^1+\ c_{14,4}-\ c_{14,5}+p\ c_{15,4}-\ c_{15,5}-\ B_{7,3}^1+\ c_{14,6}+p\ c_{14,7}-$ $c_{15,6}-\ c_{15,7}+n$
LL_3^2	r	$c_{0,1}+\ c_{1,0}+$
HH_1^4	s	$B_{4,4}^0-\ B_{4,7}^0-\ B_{6,4}^0-\ B_{6,5}^0-\ B_{7,4}^0-\ B_{6,7}^0-$
HH_2^3	s	$B_{2,2}^0+\ c_{2,2}p\ B_{3,2}^0-$
HL_1^3	s	$B_{0,5}^0-\ B_{2,4}^0+\ c_{2,4}p\ B_{2,6}^0-\ B_{3,6}^0-$
LH_1^3	s	$B_{2,0}^1+\ c_{4,0}-\ c_{4,1}+n\ c_{5,0}+n\ c_{5,1}+p\ B_{5,3}^0+\ c_{5,3}n\ B_{6,0}^0+\ c_{6,0}n\ B_{6,1}^0+\ c_{6,1}p$ $B_{7,0}^0+\ c_{7,0}p\ B_{6,2}^0-\ B_{6,3}^0+\ c_{6,3}p\ B_{7,3}^0-$
HL_2^2	s	

continued on next page

Table 5.3 – continued from previous page

set	sr	
LH_2^2	s	$B_{3,1}^0-$
LL_3^1	s	$B_{0,0}^0+\ c_{0,0}n\ B_{1,1}^0+\ c_{1,1}p$
HH_0^5	r	
HH_0^4	s	$B_{4,4}^1-\ B_{4,5}^1-\ B_{5,4}^1-\ B_{10,10}^0-\ B_{10,11}^0+\ c_{10,11}p\ B_{11,10}^0-\ B_{8,12}^0-\ B_{8,13}^0-$ $B_{8,14}^0+\ c_{8,14}p\ B_{9,14}^0-\ B_{10,12}^0-\ B_{10,13}^0-\ B_{11,12}^0+\ c_{11,12}n\ B_{5,7}^1-\ B_{6,4}^1-$ $B_{12,10}^0-\ B_{13,11}^0-\ B_{7,4}^1+\ c_{14,8}-\ c_{14,9}-\ c_{15,8}-\ c_{15,9}p\ B_{14,11}^0-\ B_{15,10}^0-$ $B_{15,11}^0-\ B_{12,12}^0+\ c_{12,12}n\ B_{12,13}^0-\ B_{13,12}^0-\ B_{12,14}^0-\ B_{12,15}^0+\ c_{12,15}p\ B_{13,14}^0-$ $B_{7,6}^1+\ c_{14,12}+n\ c_{14,13}+n\ c_{15,12}-\ c_{15,13}+p\ B_{14,14}^0-\ B_{14,15}^0-\ B_{15,15}^0-$
HL_0^4	r	$c_{3,8}-\ c_{2,14}+\ c_{4,8}-\ c_{5,8}-\ c_{6,8}-\ c_{7,8}-\ c_{6,15}+\ c_{1,8}-\ c_{2,8}+\ c_{0,15}+\ c_{1,14}-$ $c_{1,15}-\ c_{2,15}-\ c_{4,14}-\ c_{5,15}-\ c_{6,14}-\ c_{7,14}+$
LH_0^4	r	$c_{10,4}-\ c_{13,5}-$
HH_1^4	r	$c_{5,5}+\ c_{5,6}-\ c_{5,7}+\ c_{7,5}-\ c_{7,7}-\ c_{4,5}-\ c_{5,4}+\ c_{4,6}+\ c_{6,6}+\ c_{7,6}+$
HH_2^3	r	$c_{2,3}+\ c_{3,3}-$
HL_0^3	s	$B_{0,9}^0-\ B_{1,9}^0-\ B_{0,5}^1+\ c_{0,10}-\ c_{0,11}+p\ c_{1,10}-\ c_{1,11}+p\ B_{3,9}^0+\ c_{3,9}n\ B_{1,5}^1+$ $c_{2,10}+n\ c_{2,11}-\ c_{3,10}-\ c_{3,11}+n\ B_{0,6}^1+\ c_{0,12}+n\ c_{0,13}+p\ c_{1,12}+n\ c_{1,13}+n$ $B_{3,12}^0-\ B_{3,13}^0+\ c_{3,13}p\ B_{5,9}^0+\ c_{5,9}n\ B_{5,11}^0-\ B_{7,9}^0+\ c_{7,9}n\ B_{3,5}^1+\ c_{6,10}-\ c_{6,11}-$ $c_{7,10}-\ c_{7,11}p\ B_{4,15}^0-$
LH_0^3	s	$B_{4,0}^1+\ c_{8,0}+p\ c_{8,1}-\ c_{9,0}+n\ c_{9,1}-\ B_{4,1}^1+\ c_{8,2}-\ c_{8,3}+p\ c_{9,2}-\ c_{9,3}+n\ B_{5,0}^1-$ $B_{10,2}^0-\ B_{11,2}^0-\ B_{11,3}^0+\ c_{11,3}n\ B_{8,5}^0-\ B_{9,5}^0+\ c_{9,5}n\ B_{8,6}^0+\ c_{8,6}p\ B_{8,7}^0-\ B_{9,7}^0-$ $B_{5,3}^1+\ c_{10,6}-\ c_{10,7}-\ c_{11,6}+n\ c_{11,7}-\ B_{6,0}^1+\ c_{12,0}+n\ c_{12,1}-\ c_{13,0}-\ c_{13,1}-$ $B_{12,3}^0+\ c_{12,3}p\ B_{13,2}^0-\ B_{13,3}^0+\ c_{13,3}n\ B_{7,0}^1+\ c_{14,0}+p\ c_{14,1}+n\ c_{15,0}+p\ c_{15,1}-$ $B_{7,1}^1+\ c_{14,2}-\ c_{14,3}+p\ c_{15,2}-\ c_{15,3}+p\ B_{12,4}^0+\ c_{12,4}p\ B_{13,4}^0-\ B_{12,6}^0-\ B_{12,7}^0-$ $B_{13,7}^0+\ c_{13,7}p\ B_{14,4}^0-\ B_{15,4}^0-\ B_{15,5}^0+\ c_{15,5}n\ B_{14,7}^0+\ c_{14,7}p\ B_{15,6}^0+\ c_{15,6}n$
HL_1^3	r	$c_{0,7}-\ c_{2,7}+\ c_{0,4}+\ c_{0,6}+\ c_{2,5}-\ c_{3,5}-\ c_{1,4}-\ c_{1,5}+\ c_{1,6}-\ c_{1,7}-\ c_{3,4}+\ c_{3,7}-$
LH_1^3	r	$c_{4,3}+\ c_{5,2}-\ c_{7,2}-\ c_{4,2}-\ c_{7,1}-$
HL_2^2	r	$c_{1,2}-\ c_{0,2}+\ c_{0,3}+\ c_{1,3}+$
LH_2^2	r	$c_{2,1}-\ c_{2,0}-\ c_{3,0}-$
LL_3^1	r	$c_{0,1}+\ c_{1,0}+$
HH_1^3	s	$B_{4,4}^0+\ c_{4,4}p\ B_{4,7}^0-\ B_{6,4}^0-\ B_{6,5}^0-\ B_{7,4}^0+\ c_{7,4}n\ B_{6,7}^0-$
HH_2^2	s	$B_{3,2}^0-$

continued on next page

Table 5.3 – continued from previous page

set	sr	
HL_1^2	s	$B_{0,5}^0+ c_{0,5}p\ B_{2,6}^0+ c_{2,6}n\ B_{3,6}^0-$
LH_1^2	s	$B_{4,0}^0+ c_{4,0}n\ B_{6,2}^0+ c_{6,2}n\ B_{7,3}^0-$
HL_2^1	s	
LH_2^1	s	$B_{3,1}^0+ c_{3,1}p$
LL_3^0	s	
HH_0^4	r	$c_{11,11}+ c_{9,12}- c_{9,13}- c_{8,15}+ c_{9,15}- c_{11,13}- c_{12,11}- c_{13,10}- c_{14,10}- c_{13,13}-$ $c_{13,15}- c_{15,14}+$
HH_0^3	s	$B_{4,4}^1+ c_{8,8}+n\ c_{8,9}- c_{9,8}- c_{9,9}+n\ B_{4,5}^1+ c_{8,10}- c_{8,11}+p\ c_{9,10}- c_{9,11}-$ $B_{5,4}^1+ c_{10,8}- c_{10,9}+p\ c_{11,8}- c_{11,9}- B_{10,10}^0+ c_{10,10}n\ B_{11,10}^0- B_{8,12}^0- B_{8,13}^0+$ $c_{8,13}p\ B_{9,14}^0- B_{10,12}^0- B_{10,13}^0- B_{5,7}^1+ c_{10,14}- c_{10,15}+n\ c_{11,14}+p\ c_{11,15}-$ $B_{6,4}^1+ c_{12,8}- c_{12,9}- c_{13,8}+p\ c_{13,9}- B_{12,10}^0+ c_{12,10}n\ B_{13,11}^0- B_{14,8}^0- B_{14,9}^0-$ $B_{15,8}^0+ c_{15,8}n\ B_{14,11}^0+ c_{14,11}p\ B_{15,10}^0+ c_{15,10}n\ B_{15,11}^0- B_{12,13}^0+ c_{12,13}n$ $B_{13,12}^0+ c_{13,12}p\ B_{12,14}^0- B_{13,14}^0- B_{15,12}^0+ c_{15,12}p\ B_{14,14}^0+ c_{14,14}p\ B_{14,15}^0-$ $B_{15,15}^0-$
HL_0^3	r	$c_{3,8}- c_{2,14}- c_{4,8}- c_{5,8}+ c_{6,8}+ c_{7,8}- c_{6,15}- c_{1,8}+ c_{2,8}+ c_{0,15}- c_{1,14}-$ $c_{1,15}+ c_{2,15}+ c_{4,14}- c_{5,15}- c_{6,14}- c_{7,14}- c_{0,8}- c_{2,9}+ c_{0,14}+ c_{2,12}- c_{2,13}-$ $c_{3,14}+ c_{3,15}+ c_{4,9}+ c_{4,10}+ c_{4,11}+ c_{5,10}+ c_{6,9}- c_{4,12}+ c_{4,13}+ c_{5,12}+ c_{5,13}-$ $c_{5,14}+ c_{6,12}- c_{6,13}+ c_{7,12}+ c_{7,13}- c_{7,15}-$
LH_0^3	r	$c_{10,4}- c_{13,5}+ c_{10,3}- c_{8,4}- c_{9,4}- c_{9,6}+ c_{10,5}- c_{11,4}+ c_{11,5}- c_{12,2}- c_{12,5}+$ $c_{13,6}- c_{14,5}- c_{14,6}- c_{15,7}-$
HH_1^3	r	$c_{5,5}- c_{5,6}- c_{5,7}+ c_{7,5}+ c_{7,7}- c_{4,5}- c_{5,4}+ c_{4,6}+ c_{6,6}- c_{7,6}-$
HH_2^2	r	$c_{2,3}- c_{3,3}- c_{2,2}-$
HL_0^2	s	$B_{0,9}^0+ c_{0,9}p\ B_{1,9}^0+ c_{1,9}p\ B_{0,10}^0- B_{1,10}^0+ c_{1,10}n\ B_{2,11}^0+ c_{2,11}p\ B_{3,10}^0- B_{3,12}^0+$ $c_{3,12}p\ B_{5,11}^0+ c_{5,11}p\ B_{6,10}^0+ c_{6,10}p\ B_{6,11}^0- B_{7,10}^0- B_{4,15}^0+ c_{4,15}p$
LH_0^2	s	$B_{8,1}^0- B_{9,1}^0- B_{8,2}^0- B_{9,2}^0- B_{5,0}^1+ c_{10,0}- c_{10,1}+p\ c_{11,0}- c_{11,1}- B_{10,2}^0-$ $B_{11,2}^0+ c_{11,2}n\ B_{8,5}^0- B_{8,7}^0+ c_{8,7}p\ B_{9,7}^0- B_{10,6}^0- B_{10,7}^0- B_{11,7}^0- B_{12,1}^0+$ $c_{12,1}p\ B_{13,0}^0- B_{13,1}^0- B_{13,2}^0+ c_{13,2}p\ B_{15,1}^0- B_{14,2}^0+ c_{14,2}p\ B_{15,2}^0- B_{13,4}^0-$ $B_{12,6}^0- B_{12,7}^0- B_{14,4}^0- B_{15,4}^0+ c_{15,4}p$
HL_1^2	r	$c_{0,7}+ c_{2,7}- c_{0,4}- c_{0,6}- c_{2,5}- c_{3,5}- c_{1,4}+ c_{1,5}- c_{1,6}+ c_{1,7}+ c_{3,4}- c_{3,7}+$ $c_{2,4}-$

continued on next page

Table 5.3 – continued from previous page

set	sr	
LH_1^2	r	$c_{4,3}+\ c_{5,2}+\ c_{7,2}-\ c_{4,2}+\ c_{7,1}+\ c_{4,1}+\ c_{5,0}-\ c_{5,1}+\ c_{5,3}+\ c_{6,0}-\ c_{6,1}-\ c_{7,0}-$ $c_{6,3}+$
HL_2^1	r	$c_{1,2}-\ c_{0,2}+\ c_{0,3}+\ c_{1,3}-$
LH_2^1	r	$c_{2,1}+\ c_{2,0}+\ c_{3,0}-$
LL_3^0	r	$c_{0,1}-\ c_{1,0}-\ c_{0,0}-\ c_{1,1}-$
HH_1^2	s	$B_{4,7}^0+\ c_{4,7}n\ B_{6,4}^0-\ B_{6,5}^0-\ B_{6,7}^0-$
HH_2^1	s	$B_{3,2}^0-$
HL_1^1	s	$B_{3,6}^0+\ c_{3,6}n$
LH_1^1	s	$B_{7,3}^0-$
HL_2^0	s	
LH_2^0	s	
HH_0^3	r	$c_{11,11}-\ c_{9,12}+\ c_{9,13}+\ c_{8,15}-\ c_{9,15}+\ c_{11,13}+\ c_{12,11}-\ c_{13,10}-\ c_{14,10}-\ c_{13,13}-$ $c_{13,15}+\ c_{15,14}-\ c_{10,11}-\ c_{8,14}+\ c_{11,12}+\ c_{15,9}+\ c_{12,12}+\ c_{12,15}-\ c_{14,12}-$ $c_{14,13}+\ c_{15,13}+$
HH_0^2	s	$B_{8,9}^0+\ c_{8,9}p\ B_{9,8}^0+\ c_{9,8}p\ B_{8,10}^0-\ B_{9,10}^0+\ c_{9,10}p\ B_{9,11}^0-\ B_{10,8}^0-\ B_{11,8}^0-$ $B_{11,9}^0+\ c_{11,9}n\ B_{11,10}^0-\ B_{8,12}^0+\ c_{8,12}p\ B_{9,14}^0+\ c_{9,14}n\ B_{10,12}^0-\ B_{10,13}^0-$ $B_{10,14}^0+\ c_{10,14}n\ B_{11,15}^0+\ c_{11,15}n\ B_{12,8}^0-\ B_{12,9}^0-\ B_{13,9}^0-\ B_{13,11}^0+\ c_{13,11}n$ $B_{14,8}^0-\ B_{14,9}^0+\ c_{14,9}p\ B_{15,11}^0+\ c_{15,11}n\ B_{12,14}^0-\ B_{13,14}^0-\ B_{14,15}^0+\ c_{14,15}p$ $B_{15,15}^0-$
HL_0^2	r	$c_{3,8}-\ c_{2,14}-\ c_{4,8}-\ c_{5,8}+\ c_{6,8}+\ c_{7,8}+\ c_{6,15}+\ c_{1,8}+\ c_{2,8}+\ c_{0,15}+\ c_{1,14}+$ $c_{1,15}-\ c_{2,15}+\ c_{4,14}-\ c_{5,15}-\ c_{6,14}+\ c_{7,14}+\ c_{0,8}-\ c_{2,9}+\ c_{0,14}+\ c_{2,12}-\ c_{2,13}-$ $c_{3,14}-\ c_{3,15}-\ c_{4,9}+\ c_{4,10}-\ c_{4,11}+\ c_{5,10}-\ c_{6,9}+\ c_{4,12}+\ c_{4,13}-\ c_{5,12}+\ c_{5,13}-$ $c_{5,14}-\ c_{6,12}+\ c_{6,13}+\ c_{7,12}-\ c_{7,13}+\ c_{7,15}-\ c_{0,11}-\ c_{1,11}-\ c_{3,9}+\ c_{2,10}-\ c_{3,11}+$ $c_{0,12}-\ c_{0,13}-\ c_{1,12}-\ c_{1,13}+\ c_{3,13}+\ c_{5,9}+\ c_{7,9}-\ c_{7,11}+$
LH_0^2	r	$c_{10,4}-\ c_{13,5}-\ c_{10,3}+\ c_{8,4}-\ c_{9,4}-\ c_{9,6}-\ c_{10,5}-\ c_{11,4}-\ c_{11,5}+\ c_{12,2}+\ c_{12,5}-$ $c_{13,6}-\ c_{14,5}+\ c_{14,6}-\ c_{15,7}+\ c_{8,0}-\ c_{9,0}-\ c_{8,3}-\ c_{9,3}-\ c_{11,3}+\ c_{9,5}+\ c_{8,6}-$ $c_{11,6}-\ c_{12,0}-\ c_{12,3}+\ c_{13,3}+\ c_{14,0}-\ c_{14,1}-\ c_{15,0}-\ c_{14,3}-\ c_{15,3}-\ c_{12,4}-$ $c_{13,7}+\ c_{15,5}+\ c_{14,7}-\ c_{15,6}+$
HH_1^2	r	$c_{5,5}-\ c_{5,6}+\ c_{5,7}-\ c_{7,5}-\ c_{7,7}-\ c_{4,5}-\ c_{5,4}+\ c_{4,6}+\ c_{6,6}-\ c_{7,6}-\ c_{4,4}-\ c_{7,4}-$
HH_2^1	r	$c_{2,3}+\ c_{3,3}-\ c_{2,2}-$

continued on next page

Table 5.3 – continued from previous page

set	**sr**	
HL_0^1	s	$B^0_{0,10}- B^0_{3,10}- B^0_{6,11}+ c_{6,11}p\ B^0_{7,10}+ c_{7,10}n$
LH_0^1	s	$B^0_{8,1}- B^0_{9,1}+ c_{9,1}n\ B^0_{8,2}- B^0_{9,2}+ c_{9,2}p\ B^0_{10,0}- B^0_{11,0}- B^0_{11,1}+ c_{11,1}n\ B^0_{10,2}-$ $B^0_{8,5}+ c_{8,5}p\ B^0_{9,7}+ c_{9,7}n\ B^0_{10,6}- B^0_{10,7}- B^0_{11,7}+ c_{11,7}p\ B^0_{13,0}- B^0_{13,1}-$ $B^0_{15,1}- B^0_{15,2}- B^0_{13,4}+ c_{13,4}n\ B^0_{12,6}- B^0_{12,7}- B^0_{14,4}-$
HL_1^1	r	$c_{0,7}- c_{2,7}+ c_{0,4}- c_{0,6}+ c_{2,5}- c_{3,5}- c_{1,4}+ c_{1,5}- c_{1,6}- c_{1,7}+ c_{3,4}- c_{3,7}+$ $c_{2,4}- c_{0,5}+ c_{2,6}-$
LH_1^1	r	$c_{4,3}- c_{5,2}- c_{7,2}- c_{4,2}+ c_{7,1}- c_{4,1}+ c_{5,0}+ c_{5,1}- c_{5,3}- c_{6,0}- c_{6,1}- c_{7,0}+$ $c_{6,3}- c_{4,0}- c_{6,2}-$
HL_2^0	r	$c_{1,2}- c_{0,2}+ c_{0,3}+ c_{1,3}-$
LH_2^0	r	$c_{2,1}- c_{2,0}- c_{3,0}+ c_{3,1}+$
HH_1^1	s	$B^0_{6,4}- B^0_{6,5}+ c_{6,5}p\ B^0_{6,7}+ c_{6,7}n$
HH_2^0	s	$B^0_{3,2}+ c_{3,2}p$
HL_1^0	s	
LH_1^0	s	$B^0_{7,3}+ c_{7,3}p$
HH_0^2	r	$c_{11,11}+ c_{9,12}- c_{9,13}- c_{8,15}+ c_{9,15}+ c_{11,13}- c_{12,11}+ c_{13,10}+ c_{14,10}+ c_{13,13}-$ $c_{13,15}+ c_{15,14}+ c_{10,11}+ c_{8,14}- c_{11,12}- c_{15,9}+ c_{12,12}- c_{12,15}- c_{14,12}-$ $c_{14,13}- c_{15,13}- c_{8,8}- c_{9,9}- c_{8,11}+ c_{10,9}+ c_{10,10}- c_{8,13}+ c_{10,15}- c_{11,14}-$ $c_{13,8}- c_{12,10}+ c_{15,8}- c_{14,11}- c_{15,10}+ c_{12,13}+ c_{13,12}+ c_{15,12}- c_{14,14}-$
HH_0^1	s	$B^0_{8,10}+ c_{8,10}n\ B^0_{9,11}+ c_{9,11}p\ B^0_{10,8}- B^0_{11,8}- B^0_{11,10}+ c_{11,10}n\ B^0_{10,12}+ c_{10,12}p$ $B^0_{10,13}+ c_{10,13}p\ B^0_{12,8}- B^0_{12,9}+ c_{12,9}n\ B^0_{13,9}+ c_{13,9}n\ B^0_{14,8}+ c_{14,8}p\ B^0_{12,14}-$ $B^0_{13,14}+ c_{13,14}p\ B^0_{15,15}-$
HL_0^1	r	$c_{3,8}- c_{2,14}- c_{4,8}+ c_{5,8}+ c_{6,8}- c_{7,8}- c_{6,15}+ c_{1,8}- c_{2,8}- c_{0,15}- c_{1,14}-$ $c_{1,15}+ c_{2,15}+ c_{4,14}- c_{5,15}+ c_{6,14}- c_{7,14}+ c_{0,8}+ c_{2,9}- c_{0,14}+ c_{2,12}- c_{2,13}-$ $c_{3,14}+ c_{3,15}- c_{4,9}+ c_{4,10}+ c_{4,11}- c_{5,10}+ c_{6,9}- c_{4,12}- c_{4,13}- c_{5,12}+ c_{5,13}-$ $c_{5,14}- c_{6,12}+ c_{6,13}- c_{7,12}- c_{7,13}- c_{7,15}- c_{0,11}- c_{1,11}- c_{3,9}+ c_{2,10}- c_{3,11}+$ $c_{0,12}- c_{0,13}+ c_{1,12}- c_{1,13}+ c_{3,13}+ c_{5,9}+ c_{7,9}- c_{7,11}+ c_{0,9}+ c_{1,9}- c_{1,10}+$ $c_{2,11}+ c_{3,12}+ c_{5,11}- c_{6,10}+ c_{4,15}-$

continued on next page

Table 5.3 – continued from previous page

set	sr	
LH_0^1	r	$c_{10,4}- c_{13,5}+ c_{10,3}+ c_{8,4}+ c_{9,4}- c_{9,6}- c_{10,5}+ c_{11,4}- c_{11,5}- c_{12,2}- c_{12,5}-$ $c_{13,6}- c_{14,5}- c_{14,6}- c_{15,7}- c_{8,0}- c_{9,0}- c_{8,3}- c_{9,3}- c_{11,3}+ c_{9,5}- c_{8,6}-$ $c_{11,6}- c_{12,0}+ c_{12,3}+ c_{13,3}- c_{14,0}+ c_{14,1}- c_{15,0}- c_{14,3}- c_{15,3}+ c_{12,4}-$ $c_{13,7}+ c_{15,5}- c_{14,7}+ c_{15,6}+ c_{10,1}- c_{11,2}- c_{8,7}- c_{12,1}+ c_{13,2}+ c_{14,2}- c_{15,4}-$
HH_1^1	r	$c_{5,5}+ c_{5,6}+ c_{5,7}+ c_{7,5}- c_{7,7}- c_{4,5}+ c_{5,4}- c_{4,6}- c_{6,6}+ c_{7,6}- c_{4,4}- c_{7,4}+$ $c_{4,7}-$
HH_2^0	r	$c_{2,3}- c_{3,3}+ c_{2,2}+$
HL_0^0	s	$B^0_{0,10}+ c_{0,10}n\ B^0_{3,10}+ c_{3,10}n$
LH_0^0	s	$B^0_{8,1}+ c_{8,1}n\ B^0_{8,2}+ c_{8,2}p\ B^0_{10,0}- B^0_{11,0}+ c_{11,0}p\ B^0_{10,2}+ c_{10,2}p\ B^0_{10,6}+ c_{10,6}p$ $B^0_{10,7}+ c_{10,7}p\ B^0_{13,0}+ c_{13,0}n\ B^0_{13,1}- B^0_{15,1}+ c_{15,1}n\ B^0_{15,2}+ c_{15,2}p\ B^0_{12,6}+$ $c_{12,6}n\ B^0_{12,7}+ c_{12,7}n\ B^0_{14,4}+ c_{14,4}n$
HL_1^0	r	$c_{0,7}- c_{2,7}+ c_{0,4}+ c_{0,6}+ c_{2,5}+ c_{3,5}+ c_{1,4}+ c_{1,5}- c_{1,6}- c_{1,7}- c_{3,4}+ c_{3,7}+$ $c_{2,4}- c_{0,5}- c_{2,6}+ c_{3,6}-$
LH_1^0	r	$c_{4,3}- c_{5,2}- c_{7,2}- c_{4,2}- c_{7,1}- c_{4,1}+ c_{5,0}+ c_{5,1}+ c_{5,3}- c_{6,0}+ c_{6,1}+ c_{7,0}+$ $c_{6,3}+ c_{4,0}+ c_{6,2}+$
HH_1^0	s	$B^0_{6,4}+ c_{6,4}p$
HH_0^1	r	$c_{11,11}- c_{9,12}- c_{9,13}- c_{8,15}+ c_{9,15}+ c_{11,13}+ c_{12,11}+ c_{13,10}- c_{14,10}- c_{13,13}+$ $c_{13,15}- c_{15,14}- c_{10,11}- c_{8,14}+ c_{11,12}- c_{15,9}- c_{12,12}+ c_{12,15}- c_{14,12}-$ $c_{14,13}+ c_{15,13}- c_{8,8}+ c_{9,9}+ c_{8,11}- c_{10,9}+ c_{10,10}- c_{8,13}- c_{10,15}+ c_{11,14}-$ $c_{13,8}+ c_{12,10}- c_{15,8}- c_{14,11}+ c_{15,10}- c_{12,13}+ c_{13,12}- c_{15,12}+ c_{14,14}+ c_{8,9}-$ $c_{9,8}- c_{9,10}+ c_{11,9}- c_{8,12}+ c_{9,14}- c_{10,14}- c_{11,15}- c_{13,11}+ c_{14,9}+ c_{15,11}+$ $c_{14,15}-$
HH_0^0	s	$B^0_{10,8}+ c_{10,8}p\ B^0_{11,8}- B^0_{12,8}- B^0_{12,14}+ c_{12,14}n\ B^0_{15,15}+ c_{15,15}p$
HL_0^0	r	$c_{3,8}- c_{2,14}- c_{4,8}+ c_{5,8}+ c_{6,8}+ c_{7,8}- c_{6,15}+ c_{1,8}+ c_{2,8}+ c_{0,15}- c_{1,14}+$ $c_{1,15}+ c_{2,15}+ c_{4,14}- c_{5,15}- c_{6,14}+ c_{7,14}+ c_{0,8}+ c_{2,9}- c_{0,14}- c_{2,12}+ c_{2,13}+$ $c_{3,14}- c_{3,15}+ c_{4,9}- c_{4,10}+ c_{4,11}- c_{5,10}- c_{6,9}- c_{4,12}- c_{4,13}+ c_{5,12}- c_{5,13}+$ $c_{5,14}- c_{6,12}- c_{6,13}+ c_{7,12}- c_{7,13}- c_{7,15}- c_{0,11}+ c_{1,11}- c_{3,9}+ c_{2,10}+ c_{3,11}+$ $c_{0,12}+ c_{0,13}+ c_{1,12}- c_{1,13}+ c_{3,13}+ c_{5,9}+ c_{7,9}+ c_{7,11}- c_{0,9}+ c_{1,9}+ c_{1,10}+$ $c_{2,11}- c_{3,12}+ c_{5,11}- c_{6,10}- c_{4,15}+ c_{6,11}- c_{7,10}+$

continued on next page

Table 5.3 – continued from previous page

set	sr	
LH_0^0	r	$c_{10,4}+\ c_{13,5}-\ c_{10,3}-\ c_{8,4}-\ c_{9,4}+\ c_{9,6}-\ c_{10,5}-\ c_{11,4}+\ c_{11,5}-\ c_{12,2}-\ c_{12,5}-$ $c_{13,6}+\ c_{14,5}+\ c_{14,6}-\ c_{15,7}+\ c_{8,0}+\ c_{9,0}-\ c_{8,3}-\ c_{9,3}-\ c_{11,3}+\ c_{9,5}+\ c_{8,6}+$ $c_{11,6}-\ c_{12,0}-\ c_{12,3}-\ c_{13,3}+\ c_{14,0}-\ c_{14,1}-\ c_{15,0}-\ c_{14,3}+\ c_{15,3}+\ c_{12,4}+$ $c_{13,7}+\ c_{15,5}+\ c_{14,7}+\ c_{15,6}+\ c_{10,1}-\ c_{11,2}-\ c_{8,7}-\ c_{12,1}-\ c_{13,2}+\ c_{14,2}-\ c_{15,4}-$ $c_{9,1}+\ c_{9,2}-\ c_{11,1}-\ c_{8,5}+\ c_{9,7}+\ c_{11,7}-\ c_{13,4}-$
HH_1^0	r	$c_{5,5}-\ c_{5,6}+\ c_{5,7}-\ c_{7,5}+\ c_{7,7}+\ c_{4,5}+\ c_{5,4}-\ c_{4,6}+\ c_{6,6}+\ c_{7,6}-\ c_{4,4}-\ c_{7,4}+$ $c_{4,7}-\ c_{6,5}+\ c_{6,7}+$
HH_0^0	r	$c_{11,11}+\ c_{9,12}+\ c_{9,13}+\ c_{8,15}+\ c_{9,15}-\ c_{11,13}-\ c_{12,11}-\ c_{13,10}+\ c_{14,10}-\ c_{13,13}-$ $c_{13,15}+\ c_{15,14}+\ c_{10,11}-\ c_{8,14}-\ c_{11,12}-\ c_{15,9}-\ c_{12,12}+\ c_{12,15}-\ c_{14,12}+$ $c_{14,13}-\ c_{15,13}-\ c_{8,8}-\ c_{9,9}-\ c_{8,11}+\ c_{10,9}+\ c_{10,10}+\ c_{8,13}+\ c_{10,15}-\ c_{11,14}+$ $c_{13,8}-\ c_{12,10}-\ c_{15,8}-\ c_{14,11}-\ c_{15,10}+\ c_{12,13}-\ c_{13,12}-\ c_{15,12}-\ c_{14,14}-\ c_{8,9}-$ $c_{9,8}+\ c_{9,10}-\ c_{11,9}+\ c_{8,12}-\ c_{9,14}+\ c_{10,14}-\ c_{11,15}+\ c_{13,11}+\ c_{14,9}-\ c_{15,11}-$ $c_{14,15}+\ c_{8,10}-\ c_{9,11}-\ c_{11,10}+\ c_{10,12}-\ c_{10,13}-\ c_{12,9}-\ c_{13,9}-\ c_{14,8}-\ c_{13,14}+$

5.8 Summary

The DEBT algorithm is a set partitioning algorithm which takes into account both the gains of the DWT decomposition and the distortion reduction calculated from a PDF fitted to the transform coefficients in order to create a priority list according to which the encoding should yield a rate-distortion optimized stream.

It can be used with any kind of DWT and allows for lossless encoding when the transform is also lossless, i.e., maps integers into integers. It is also bit oriented and its output can be truncated at any point yielding the "best" (according to the MSE error measure) approximation to the original image for the resulting prefix size.

It does not rely on a final entropy coding pass and is easily parallelizable specially when using only blocks. In fact, the use of trees do not actually improve compression performance and its actual value should be investigated further in order to find out how to take advantage of the correlation across scales.

Chapter 6

Region Of Interest

Under very low bandwidth constraints, there simply is not much that can be done in order to ensure a sufficient quality reconstruction of an image, irrespective of the algorithm used.

Most of the time, however, the user is more interested in only a small fraction of the whole image, which is called a Region Of Interest (ROI). The ROI could be either a single simple geometric shape or complex disjoint areas of an image.

This chapter examines the techniques that can be applied to the compression phase once the ROI has been selected. The ROI is an area of the image that should be prioritized during compression and transmission and will be referred to as the "foreground" (FG) in contrast to the remaining (complementary) area of the image, called the "background" (BG). The idea is to subtract bits from the background and add them to the foreground, keeping a constant bit budget while enhancing the foreground and worsening the background.

Also important is how much to improve the foreground at the expense of the background which will lead to a choice of bitplane masks that share the common lower bitplanes of both foreground and background.

6.1 Introduction

The main ROI coding technique revolves around artificially increasing the distortion reduction contribution of the foreground coefficients in order to force the bit allocation algorithm to assign a higher number of bits to it. The ROI information must be transmitted to the decoder, either explicitly of implicitly, so that it can properly decode both foreground and background coefficients.

Bitplane encoding makes it straightforward to progressively encode foreground and background coefficients, allowing for efficient lossy or lossless image encoding with or without ROI information.

At first glance, a general solution would be to break the image into 2 independent foreground and background images, compress, and transmit them in order. This is equivalent to creating a single progressive image where the foreground coefficients are shifted in such a way that they are all above the highest background bitplane, depicted in Figure 6.1 and referred to as the "maxshift" method. This would have the "advantage" of not requiring explicit information about the foreground bounding boxes and, while this would certainly give priority to the foreground, it would also use most of the initial bandwidth with the noisy lower bitplanes of the foreground coefficients which are not really needed at this point and wastes limited bandwidth with "unnecessary" data instead of using it to convey some background information.

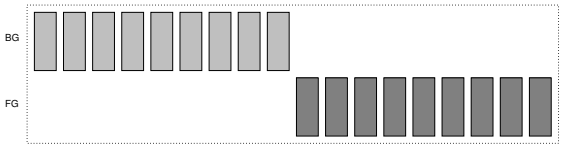


FIGURE 6.1: Maxshift ROI

It should be noted that, while the ROI map is not strictly necessary in this case, the encoder must send the insignificant state of the background coefficients for each foreground exclusive bitplane in case the decoder does not have the ROI map. If the decoder had the ROI map, such information would not be necessary because both encoder and decoder would know that such coefficients would never be significant in a foreground exclusive bitplane and, therefore, their insignificance do not need to be conveyed.

6.2 ROI types

There are essentially 2 types of ROI, depending if the decoder has or not a priori knowledge of which coefficients are in the foreground and in the background, i.e., depending if the encoder sends or not the ROI map:

- Map ROI - The ROI map is encoded as overhead information and conveyed to the decoder. This allows for any combination of exclusive and shared foreground and background bitplanes and the efficient encoding of information based on the position of the coefficients and the kind of bitplane, i.e., foreground exclusive, background exclusive, or shared. However, the ROI map should be able to be encoded with a small number of bits in order for it to be effective.
- Bitplane ROI - The decoder has no knowledge of the ROI geometry which allows for complex shaped ROIs. While this has the advantage of not requiring any prior ROI map information, the bitplanes must be arranged in such a way that all bitplanes are either foreground or background exclusive, except for a number of initial consecutive bitplanes (LSBs), which can be shared. Also, the decoder only knows if coefficients belong to the foreground or background when it receives information about its first significant bitplane (MSB), either foreground exclusive, background exclusive, or shared, in which case there is no distinction between foreground and background coefficients. This means that information regarding insignificant coefficients must be conveyed for all bitplanes.

It is important to note that, as long as the ROI map can be encoded with a small number of bits, it is generally advantageous to convey it, irrespective of the chosen bitplane arrangement, including the ones used for the Bitplane ROI. Generally, both encoding and decoding with map ROI consume more system resources and take longer in exchange for better compression ratio, specially at low rates.

6.2.1 Map ROI

Usually, if the ROI map can be described by a small number of simple geometric shapes, e.g., rectangles and/or ellipses, these shapes can be efficiently conveyed to the decoder

using a small number of bits and, in this case, both the encoder and decoder can operate with knowledge of the exact positions of each foreground and background coefficient.

This allows for a very efficient blending of foreground and background coefficients, which is only possible with knowledge of the ROI geometry, known as "scaling" which is depicted in Figure 6.2.

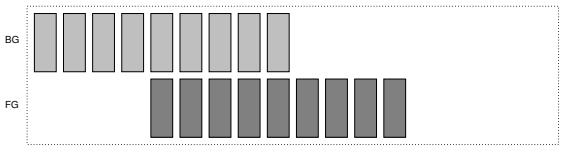


FIGURE 6.2: Scaling ROI

In the scaling method, foreground coefficients are scaled by a constant amount, usually a power of 2 so that their bitplanes are simply shifted by the base 2 logarithm of the scaling factor. The scaling depicted in Figure 6.2 can only be implemented when the decoder has knowledge of the ROI map otherwise it would not know if a coefficient which is significant at a shared bitplane is a foreground or a background coefficient.

The ROI map can be used with any any combination of exclusive and shared bitplanes, including the maxshift method, allowing for a more efficient encoding of foreground and background information on each respective exclusive bitplane, as long as the overhead necessary to transmit the map is smaller than the number of bits used to convey the insignificance of coefficients in opposite exclusive bitplanes.

In addition, it may prove advantageous to use a nonlinear scaling method in certain situations in order to better blend foreground and background regions, as depicted in Figure 6.3.

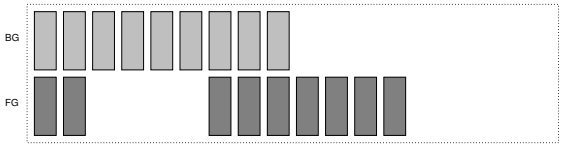


FIGURE 6.3: Nonlinear Scaling ROI

In this case, only the foreground coefficients larger than a threshold have their top bitplanes shifted by a certain number of bitplanes. This allows for the same number of bitplanes as the scaling ROI shown in Figure 6.2 but would only encode the usually large number of coefficients which are significant at the last foreground bitplanes together

with the same number of background ones, allowing for better background refinement in exchange for very detailed foreground refinement.

6.2.2 Bitplane ROI

If the ROI map overhead is too large, a ROI method that relies on exclusive bitplanes for foreground and background bitplanes can be used that mimics the way the scaling method works. Figure 6.4 depicts a bitplane ROI method that tries to, just like the scaling method, progressively blend the refinement of the foreground coefficients with the background but does not need a ROI map.

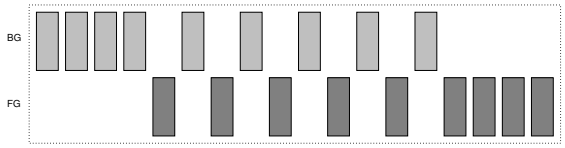


FIGURE 6.4: Exclusive ROI

The total number of bitplanes in this case is the same as in the maxshift method and other combinations of foreground and background bitplanes are also allowed. It should be noted that, just like any ROI method that does not use a ROI map, insignificance information must be sent for all coefficients at all bitplanes and this information increases as the bitplane decreases, reflecting the distribution of small valued coefficients.

A more general approach to the bitplane ROI method without the need of a ROI map and which avoids the high overhead of sending the insignificance information for the lowest foreground exclusive bitplanes is to allow, in addition to the higher exclusive foreground and background bitplanes, a number of common lower bitplanes (shared), as show in Figure 6.5.

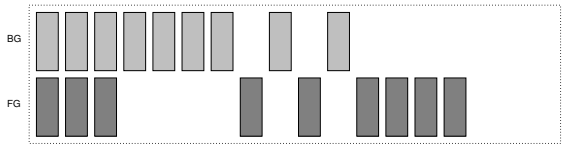


FIGURE 6.5: Bitplane ROI

In this case, the total number of bitplanes is decreased by the number of shared bitplanes and the decoder does not need to worry about coefficients which are significant

in these bitplanes because, in this case, there is no distinction between foreground and background coefficients.

Using the same reasoning, a number of shared lower bitplanes could also be used with the maxshift method resulting in the method depicted in Figure 6.6 consisting of only shifting the top bitplanes of those coefficients of the foreground that are greater than or equal to a certain threshold. This would have the advantage of using the same number of bitplanes as the scaling method and would also encode the top foreground bitplanes first, but would not refine them anymore until the corresponding number of background bitplanes have been encoded.

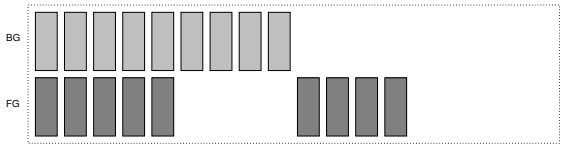


FIGURE 6.6: Top Shift ROI

Needless to say, all these bitplane configurations can also be used with the ROI map and, in this case, there is no need to convey any information regarding insignificance of coefficients which do not match their respective exclusive bitplanes, allowing for a more efficient coding, as long as the ROI map is small.

6.3 Examples

The "lena" image (512×512 pixels) was used with a ROI selected to be a circle centered at her right eye at coordinates (265,265) with a diameter of 49 pixels (inscribed in a square with its top left corner at coordinates (241,241) and bottom right corner at coordinates (289,289)), shown in Figure 6.7.

Using 6 decomposition levels, the actual deinterleaved map calculated from the given ROI and used to determine the foreground and background bits is shown in Figure 6.8 where the dark areas represent the foreground pixels and the lighter larger areas the background pixels. In calculating the map, any position of a coarser subband which had at least one final coefficient in the ROI area was marked as also being in the map.

Under these conditions, using the ibior13x7 integer DWT with 6 decomposition levels and assuming a laplacian PDF for the coefficients' distribution, the measured number

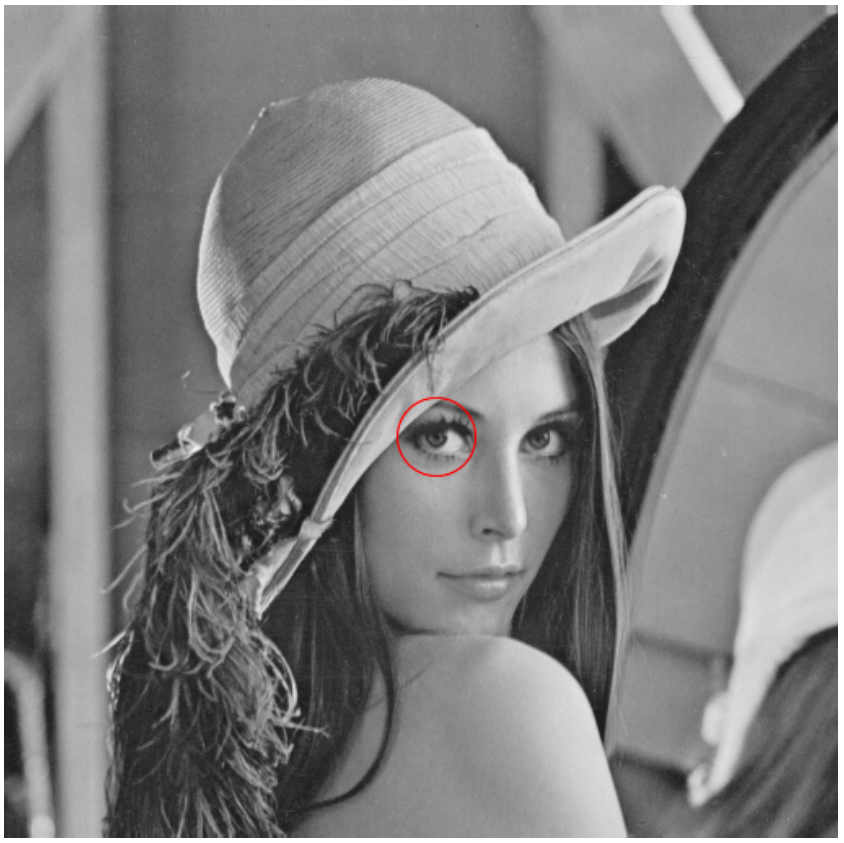


FIGURE 6.7: Lena with ROI

of bitplanes was 8 for the background and 7 for the foreground and a bitplane mask was requested in such a way that the first 4 bitplanes of the foreground were to be above the highest background bitplane with 2 shared lower bitplanes. The resulting masks are depicted in Figure 6.9 for both the bitplane (left) and scale (right) ROI methods.

The 2 masks depicted in Figure 6.9 were used with the selected ROI and, while the scale mask must have the ROI map available to the decoder, the bitplane mask does not require it but may benefit in case the decoder has the ROI map. Therefore, Figure 6.10 shows a comparison between the bitplane ROI without map (first column), bitplane ROI with map (second column) and scale ROI with map (third column).

It can be easily seen that, in all cases and for all sizes, the first 4 foreground bitplanes have been completely conveyed because there is some background information for all of them.

In this case, the ROI is composed of a simple geometric shape (circle) which can be sent as overhead with little cost and the second column depicts the use of the same bitplane

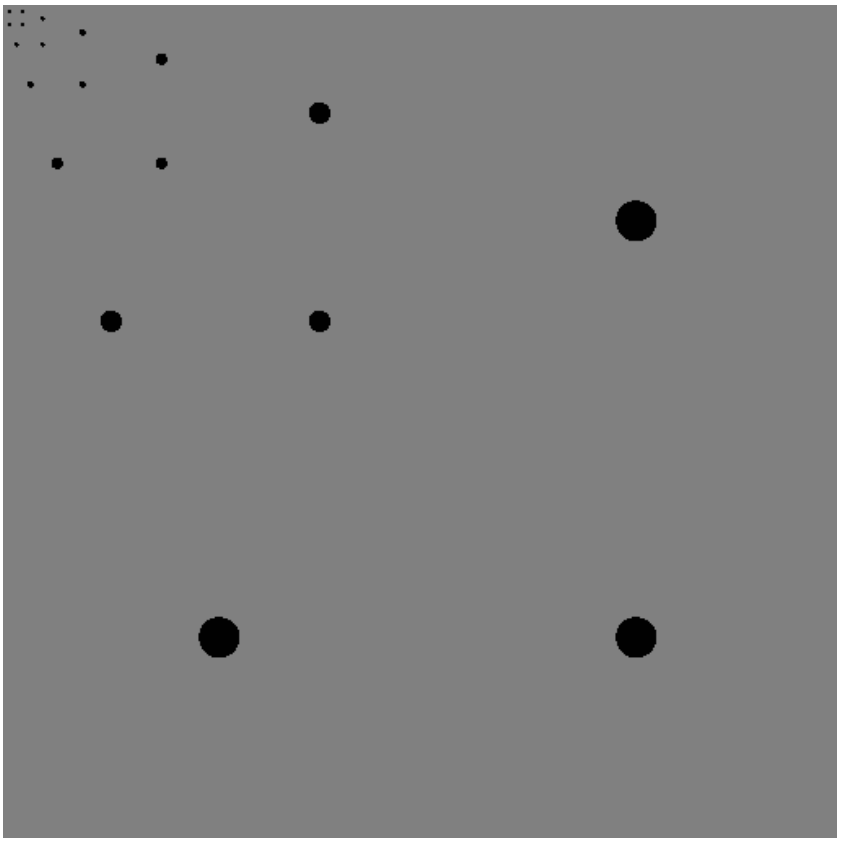


FIGURE 6.8: Lena ROI subband map

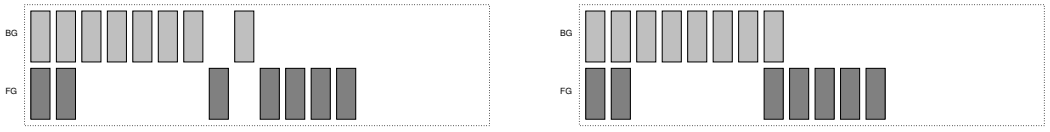


FIGURE 6.9: Bitplane (left) and Scale (right) Lena ROI masks

mask of the first column but also encodes the ROI map as overhead. The third column uses a scaling mask which necessarily implies the knowledge of the ROI map, which results in 12 bitplanes instead of 13 bitplanes of the first and second columns, as can be seen from Figure 6.9.

It is quite clear from Figure 6.10 that, in this case, sending the ROI map as overhead does pay off, resulting in better usage of the available bit budget, specially at very low bitrates. This can also be confirmed objectively by measuring the PSNR of the resulting images, shown in Table 6.1, where it can be confirmed that the scale ROI (third column) is superior to the bitmap ROI (bitplane mask with map) which, in turn, is superior to the bitplane ROI (bitplane mask without map).

FIGURE 6.10: Bit, Map, and Scale Lena ROI at 250, 500, 1000, and 2000 bytes

6.4 Summary

The use of Region Of Interest (ROI) allows for the high quality transmission of small areas of an image with its appropriate context, i.e., with some valuable information about its surroundings.

Depending on the complexity of the ROI geometry, it may be advantageous to encode

TABLE 6.1: ROI PSNR comparison (db)

Size	PSNR (db)		
(bytes)	Bit	Map	Scale
250	15.58	17.25	18.91
500	17.44	19.61	21.49
1000	21.72	23.38	23.83
2000	24.88	26.37	26.45

it and prefix it as overhead allowing for better use of the available bit budget. Otherwise, the ROI information can be conveyed by using a combination of foreground and background exclusive bitplanes, coupled with an optional number of common LSB bitplanes.

The use of common LSB bitplanes allows for the trade-off between the usually unnecessary and expensive extreme detailing of the foreground coefficients by allocating more bits for the background at this point and delaying the very detailed information to be jointly encoded at a later point.

Chapter 7

Case Studies and Results

This chapter presents, initially, benchmarks associated with the paraline algorithm and presents results of a comparison done with a publicly available implementation, followed by results and case studies for the proposed methodology and its implementation for extremely low bandwidth communications channels associated with classical images, high resolution images, and underwater images.The presented results validate the initial assumptions of both the algorithm and its implementation.

7.1 Paraline Algorithm

In this section, an older and cheaper ARM based SBC is benchmarked and a comparison with an available DWT library for the x86 platform is done which shows the performance advantage of the paraline algorithm.

Due to the lack of readily available optimized implementations for the ARM platform, no comparisons were done for it but, as stated in the chapter 3, as long as our paraline implementation is able to saturate the memory bus on this platform then comparisons should be made on the basis of implementation quality (efficiency) alone for algorithms with the same access pattern. A more useful comparison for this platform, which exhibits different available memory bandwidths depending on the memory access pattern and number of threads, would be with algorithms which possess a totally different memory access pattern.

7.1.1 32-bit ARM SBC

In order to test the paraline algorithm on a different platform than the one used in chapter 3, we have also benchmarked our implementation on an older version of the Raspberry Pi SBC. The Rasperry Pi 2 Model B (RPi2B) is based on a Broadcom BCM2836 SoC (quad core ARM Cortex-A7 with 512KB L2 cache) operating at 1 GHz with 1 GB or DDR2 RAM (32-bit single channel bus) at 500MHz (overclocked values) and is capable of running 32-bit software only.

The results obtained using our modified `stream-inlift` benchmark for the RPi2B are shown in table 7.1.

TABLE 7.1: RPi2B stream-inlift benchmark results (GB/s)

		Threads			
Kernel	MOP	1	2	3	4
Incopy	2	1.84	2.06	1.98	1.91
Copy	3	3.73	3.72	3.31	3.07
Scale	3	1.93	2.61	2.76	2.85
Add	4	1.24	2.20	2.78	2.99
Triad	4	1.03	1.95	2.49	2.88
Inlift	4	0.67	1.23	1.54	1.83
Lift	5	0.86	1.59	1.88	2.17

Table 7.1 shows that, just like the RPi3B (Table 3.5), there is a wide variation of the available bandwidth depending on the kernel (memory access pattern) and the number of threads for the RPi2B. Closer inspection reveals that the RPi2B uses the same single memory chip as the RPi3B, i.e., also uses a single 256Mb×32 DDR2 memory chip operating at 500MHz (overclocked) and tells us that the memory performance of both SBCs should be similar even though there is a large variation in processor speed. In fact, because the RPi3B uses a more efficient and higher clocked CPU than the RPi2B and their memory subsystem uses the same components clocked at the same frequencies, the performance gap between processor and memory should be higher on the RPi3B when compared to the RPi2B which makes it more likely to saturate the memory bus if at all.

Table 7.2 shows the running times in μs for the ibior5x3, ibior13x7, and icdf9x7 for the RPi2B from 1 to 4 threads.

TABLE 7.2: RPi2B lnaive time (μs)

	ibior5x3 (5 MOPs)				ibior13x7 (5 MOPs)				icdf9x7 (8 MOPs)			
Size	1	2	3	4	1	2	3	4	1	2	3	4
960×540	5550	4068	3419	3149	8644	5440	4197	3608	12449	8255	6368	5547
1280×720	10841	7223	5831	5538	16574	9825	7263	6213	24102	14808	11111	9467
1920×1080	24644	15837	12775	11715	38411	21347	15504	12969	54199	32117	23433	19820
2880×1620	55717	34596	27950	25524	88972	47615	34106	28162	122904	70046	50586	42765
3840×2160	98351	60410	49250	44849	157250	83058	59369	49117	218177	122839	88771	75085

It can be seen that our implementation gets close to saturating the memory bus for the RPi2B also and, comparing Tables 7.2 and 3.6, it can be seen that the values obtained using this older SBC get very close to the ones obtained with the newer faster RPi3B, which is expected due to their memory subsystems having the same components operating at the same frequency.

Table 7.3 shows the timings in μs for the paraline algorithm from 1 to 4 threads for the RPi2B, where it can be seen that our ibior5x3 paraline implementation saturates its memory bus for 4 threads. In fact comparing Tables 7.3 and 3.7, it can be seen that the final absolute timings for the ibior5x3 DWT are quite close for both the RPi2B and RPi3B, once again confirming the memory model used to compared memory bound algorithms.

TABLE 7.3: RPi2B paraline time (μs)

	ibior5x3 (2 MOPs)				ibior13x7 (2 MOPs)				icdf9x7 (2 MOPs)			
Size	1	2	3	4	1	2	3	4	1	2	3	4
960×540	2347	1739	1412	1326	5079	3153	2370	1985	6447	3841	2752	2321
1280×720	4409	2908	2324	2189	9178	5466	4030	3205	11638	6496	4585	3733
1920×1080	9821	6040	4792	4562	21483	11965	8479	6779	26042	14067	9778	7723
2880×1620	22274	12934	10180	9692	49226	26121	18681	14852	59919	31054	21323	16721
3840×2160	40192	22960	17855	17175	89753	46720	33225	26188	108274	55685	38062	29741

It is easy to see that the RPi3B is faster than the RPi2B but, as both have similar memory subsystems, whenever the RPi2B is capable of saturating the memory bus bandwidth, it approaches the timings of the RPi3B, as can be seen for the ibior5x3 with 4 threads paraline timings for both, given in tables 7.3 and 3.7.

Finally we have also timed our lnaive and paraline floating point implementations of the cdf9x7 DWT on the RPi2B. It should be emphasized that the executables for both the RPi2B and RPi3B are the same.

TABLE 7.4: RPi2 lnaive cdf9x7 (9 MOPs) time (μs)

Size	Threads			
	1	2	3	4
960×540	27583	17429	13390	11668
1280×720	50376	30557	22733	19797
1920×1080	111676	65542	49201	42028
2880×1620	256615	147610	109432	94077
3840×2160	454930	258015	193490	164891

TABLE 7.5: RPi2 paraline cdf9x7 (2 MOPs) time (μs)

Size	Threads			
	1	2	3	4
960×540	12155	7312	5129	4192
1280×720	22004	12894	8854	7052
1920×1080	50531	28698	19781	15640
2880×1620	115668	64902	44610	34830
3840×2160	211557	118427	81014	62871

Tables 7.4 and 7.5 show the timings for our lnaive and paraline cdf9x7 implementations, respectively, for the RPi2B. When comparing these with their respective RPi3B Tables 3.9 and 3.10, it can be seen that the RPi2B lnaive timings approach the RPi3B timings which means that we are close to saturating the RPi2B with our lnaive implementation.

7.1.2 64-bit x86 Dual Memory Channel Comparison

All DWT algorithms described in chapter 3 work inplace, do not use any auxiliary memory, and maintain the interleaved state of the original matrix. In normal use, however, an N-level dyadic decomposition of a matrix requires that an originally interleaved matrix gets replaced by its deinterleaved transformation, which then has its low pass band serving as the next decomposition's interleaved matrix. In this scenario, all inplace transform algorithms will have to do a deinterleaving operation on the transformed matrix, for each decomposition level, which can be done inplace and requires an extra 2 memory operations and 1 line's worth of auxiliary memory.

A comparison was made with a currently available DWT library (libdwt) [47][55][57] which has an inplace transform but also has a version that deinterleaves the resulting

transform matrix inplace. However, libdwt only has optimized vector code for the popular floating point CDF-9/7 DWT [46] so the comparison was made using this transform.

The CDF-9/7 has 2 lifting steps and a final low and high pass scaling and the single precision floating point version was used for comparison. In summary, our lnaive implementation for the CDF-9/7 does 9 memory operations (2 for the lines, 3 for the first lifting step, 3 for the combined last lifting step and low pass scaling, and 1 for the high pass scaling) when the matrix is kept in its current interleaved state and 10 memory operations (2 for the lines, 3 for the first lifting step, 3 for the last lifting step, and 2 for the combined deinterleaving, low, and high pass scaling) when deinterleaving the transformed matrix (our inplace deinterleaving algorithm is single threaded and uses one line of coefficients of auxiliary storage).

The paraline algorithm, as stated before, uses a fixed number of 2 memory operations for the CDF-9/7 and, in case of deinterleaving, 4 memory operations (2 for the inplace transform with no scaling and 2 for the combined deinterleaving, low, and high pass scaling).

The algorithms were compared using the absolute bandwidth saturation index S_p, given in 3.4, noting that $c = 4$ in this case due to the use of 4 byte single precision floating point coefficients. Table 7.6 shows the results for 1, 4, and 8 threads for both interleaved and deinterleaved versions of the libdwt, lnaive, and paraline algorithms while Figure 7.1 shows the results for the CDF-9/7 deinterleaved case in ns/pixel.

TABLE 7.6: Bandwidth Saturation Index - S_p - CDF-9/7

	interleaved									deinterleaved								
	libdwt (max=n/a)			lnaive (max=2/9)			paraline (max=1)			libdwt (max=n/a)			lnaive (max=1/5)			paraline (max=1/2)		
size	1	4	8	1	4	8	1	4	8	1	4	8	1	4	8	1	4	8
3840×2160	0.09	n/a	n/a	0.16	0.21	0.21	0.27	0.73	0.81	0.04	0.16	0.20	0.15	0.19	0.18	0.21	0.41	0.42
5760×3240	0.09	n/a	n/a	0.16	0.21	0.21	0.27	0.75	0.83	0.04	0.16	0.18	0.15	0.19	0.18	0.21	0.40	0.42
7680×4320	0.09	n/a	n/a	0.16	0.21	0.21	0.27	0.74	0.83	0.03	0.06	0.06	0.15	0.19	0.18	0.20	0.40	0.41
11520×6480	0.09	n/a	n/a	0.16	0.21	0.21	0.27	0.75	0.85	0.03	0.11	0.10	0.15	0.19	0.18	0.20	0.40	0.42
15360×8640	0.09	n/a	n/a	0.16	0.21	0.21	0.25	0.76	0.85	0.02	0.04	0.04	0.15	0.19	0.18	0.19	0.41	0.42

As can be seen from Table 7.6 and Figure 7.1, the paraline algorithm is vastly superior to both libdwt and lnaive, easily surpassing them with a single thread for images that don't fit in the cache. With 8 threads, our implementation gets close to saturating the memory bandwidth for the CDF-9/7 DWT, reaching 85% of the possible bandwidth for this system. As it is based on compiler intrinsics using 128-bit vectors, there is still room for improvement by code tuning and by using the newer Intel AVX extensions (256-bit

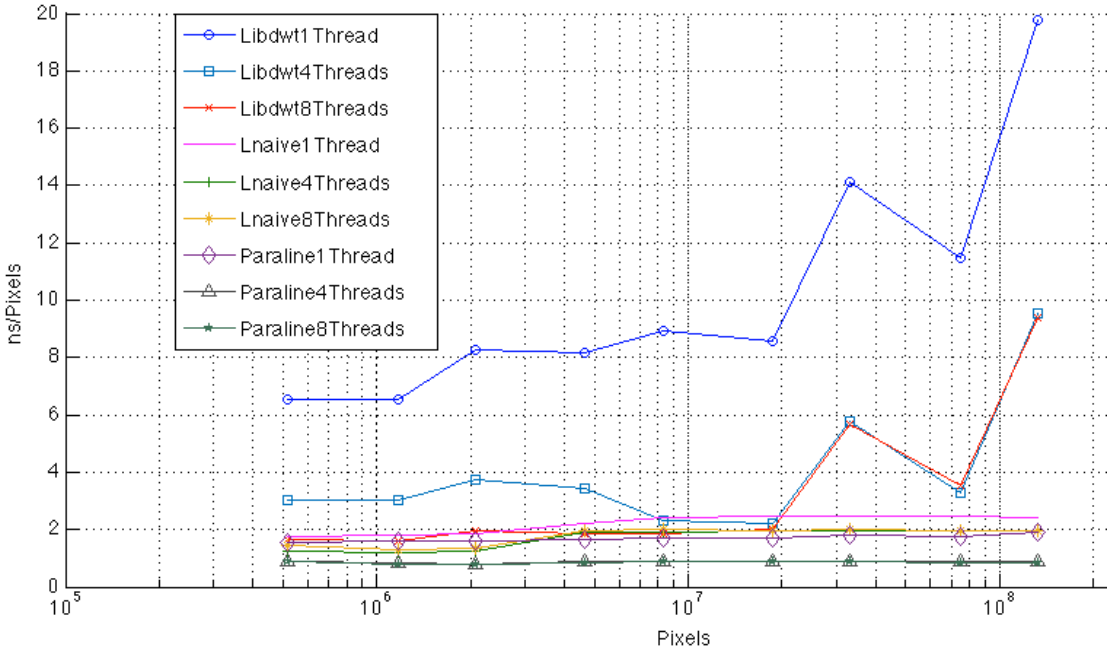


FIGURE 7.1: CDF-9/7 ns/pixel×pixel: paraline, lnaive and libdwt (deinterleaved)

vectors) which should allow it to reach minimum running time even for this DWT on this system.

As can be verified from Figure 7.1, the paraline algorithm possesses an almost constant execution time per pixel, independent of the image size, for all tested image sizes. The lnaive implementation clearly transitions from a stable level for images that fit into the cache to another level for images that do not, as expected. The libdwt algorithm benefits greatly from SMT for images smaller than a certain threshold but is, in general, inferior to the lnaive algorithm, specially for very large images.

7.2 Classic Images

This section uses classical test images like Lena, Barbara, Peppers, etc, and compare the DEBT algorithm with the state-of-the-art JPEG-2000 [23] algorithm for various bit rates. It also shows the differences between using a fitted EPD PDF or a standard laplace PDF using the DEBT algorithm with a CDF-9/7 $\langle 1, 2\rangle$ DWT, and also compare them with an uniform PDF and with an uniform PDF using the "orthogonal" subband ordering, i.e., unitary subband gains. It should be clear that for extremely low rates, the 2 extra bytes used in the header to convey the shape parameter and standard deviation

may make the compressed stream perform a little worse than the one with a predefined PDF which mean that it is important that, once the parameters are defined, an actual implementation should strive to minimize the header size while still maintaining a minimum flexibility as to be usable in different contexts.

Even though the focus was not on image compression itself but on the reordering of significant and refinement bits in order to maximize the rate distortion ratio at any truncation point, it should be emphasized that the DEBT algorithm is an order of magnitude faster than JPEG-2000 [23] (comparing our implementation of the DEBT algorithm with the jasper implementation for the JPEG-2000 [23]) and is competitive with the old JPEG standard as far as compression speed is compared.

In the next subsections, the classical images "airplane" (Figure 7.2), "baboon" (Figure 7.6), "barbara" (Figure 7.10), "lena" (Figure 7.14), "peppers" (Figure 7.18), and "sailboat" (Figure 7.22) were used in order to compare the difference in the rate distortion curves using the DEBT algorithm with a predefined laplace PDF and with an EPD PDF, which are shown in Figures 7.3, 7.7, 7.11, 7.15, 7.19, and 7.23, respectively.

The same images are also compared using the lossless ibior13x7 DWT using an EPD, a laplace, and an uniform PDF. In fact, the uniform PDF was also plotted when using the "orthogonal" ordering of the coefficients (Figure 5.1) in order to show the results that would be obtained using a standard set partitioning algorithm with a non-energy preserving DWT.

Finally, a comparison of the DEBT algorithm with an EPD PDF against the JPEG-2000 [23] algorithm is presented for each image from 1:1000 to 1:100 compression ratios in increments of 1:1000 and the results are shown in tables 7.7, 7.8, 7.9, 7.10, 7.11, and 7.12, respectively.

All images are 512×512 grayscale images and the DEBT algorithm was used with the CDF-9/7 DWT with $\langle 1, 2 \rangle$ normalization, 6 levels of decomposition, and only blocks in order to graph the rate distortion curves and to produce the comparison tables against JPEG-2000 [23]. Both rate distortion curves for each image were obtained with the DEBT algorithm using a predefined uniform PDF for the DC subband (LL subband), however, the curve on the left used a predefined laplace PDF while the curve on the right used an EPD PDF for the AC subbands. The only difference between the two is

the reordering of the coefficients according to the prioritization list $\mathcal{RDL}$ given by the possibly non-laplacian resulting PDF.

The rate distortion graphs presented were plotted using a point for every element of the rate distortion list $\mathcal{RDL}$, e.g., for $N = 6$ decomposition levels and $B = 9$ bitplanes there are $(2B - 1) + (3N)B(B + 1)/2 = 827$ images which were decoded at the end of each list element and had its PSNR evaluated and plotted when using an uniform PDF for the DC subband (LL) and a non-laplacian EPD PDF for the AC subbands, or $(3N + 1)(2B - 1) = 323$ images when using an uniform PDF for the DC subband (LL) and a laplace PDF for the AC subbands.

7.2.1 Airplane

FIGURE 7.2: Airplane

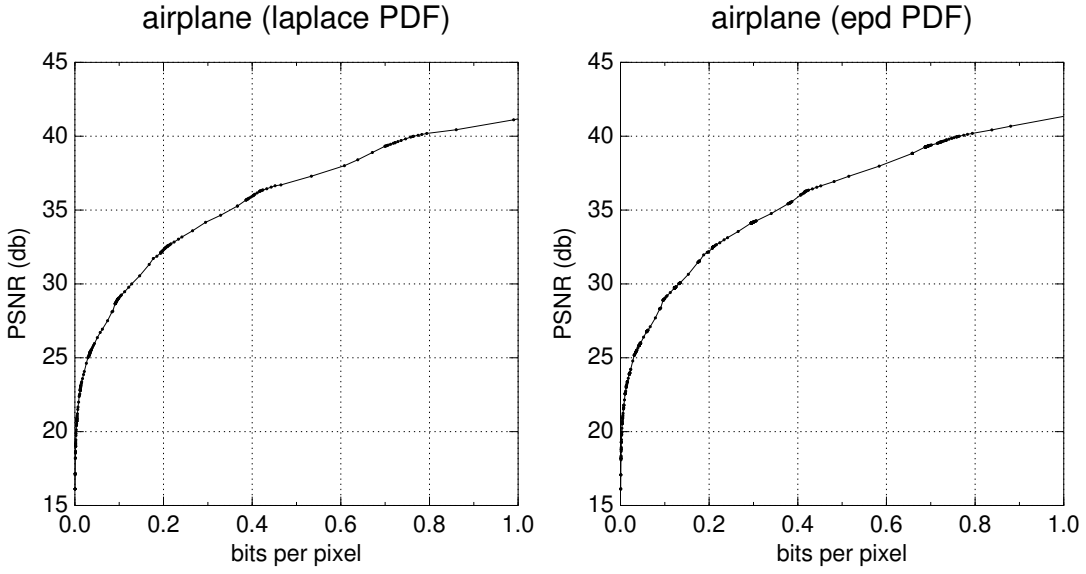


FIGURE 7.3: Airplane rate-distortion curve ($\bar{s} = 0.35$, $\bar{\sigma} = 7.66083$)

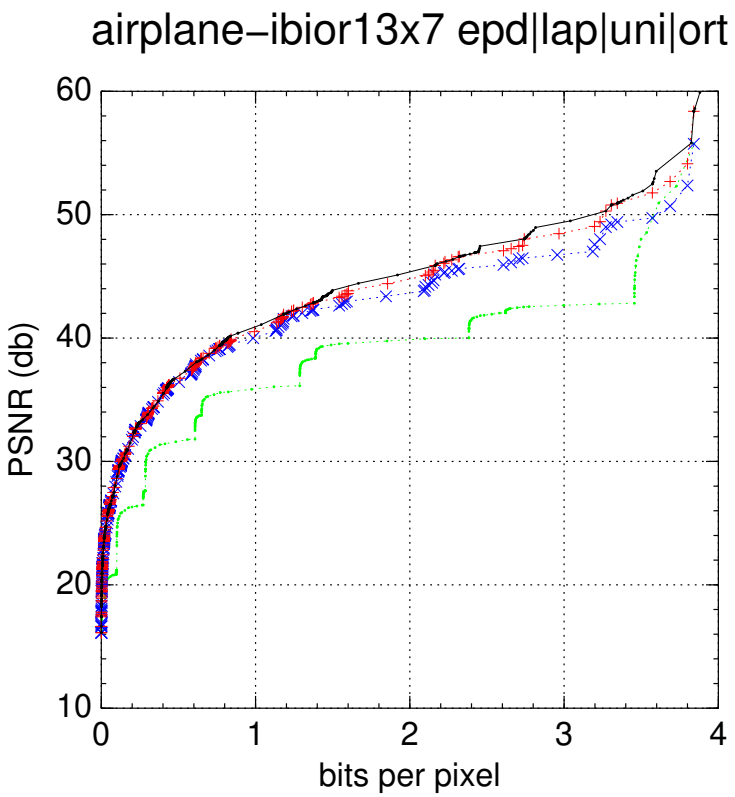


FIGURE 7.4: Airplane rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

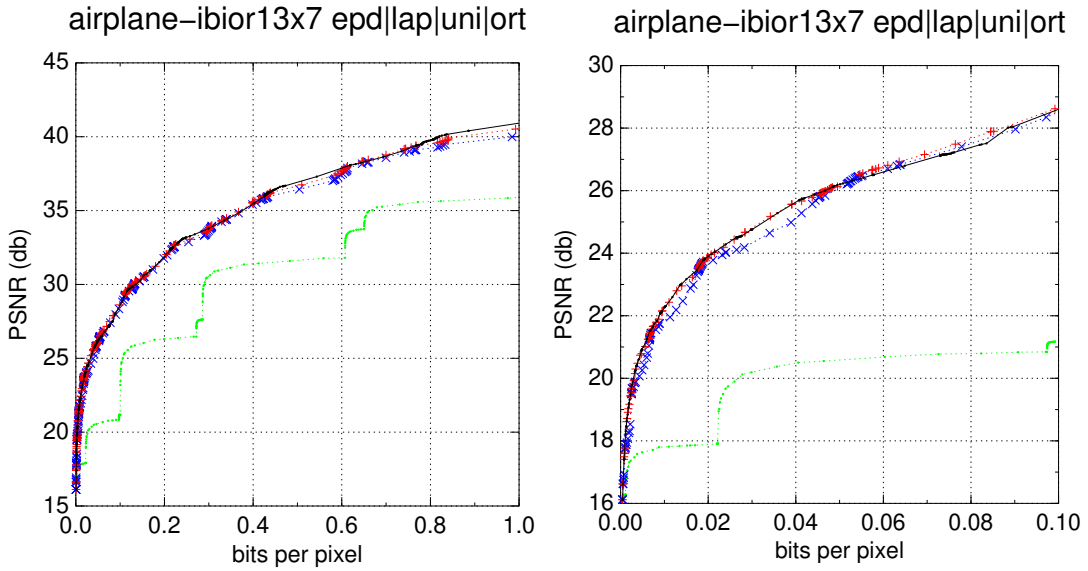


FIGURE 7.5: Low bit rate zoom for the airplane rate-distortion curve

7.2.2 Baboon

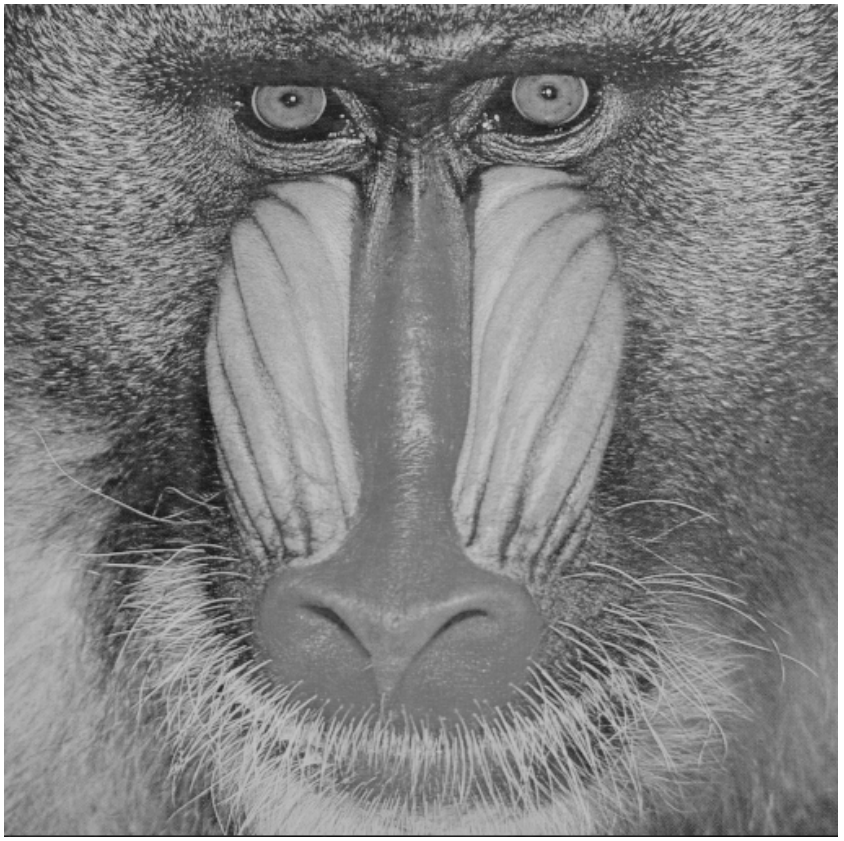


FIGURE 7.6: Baboon

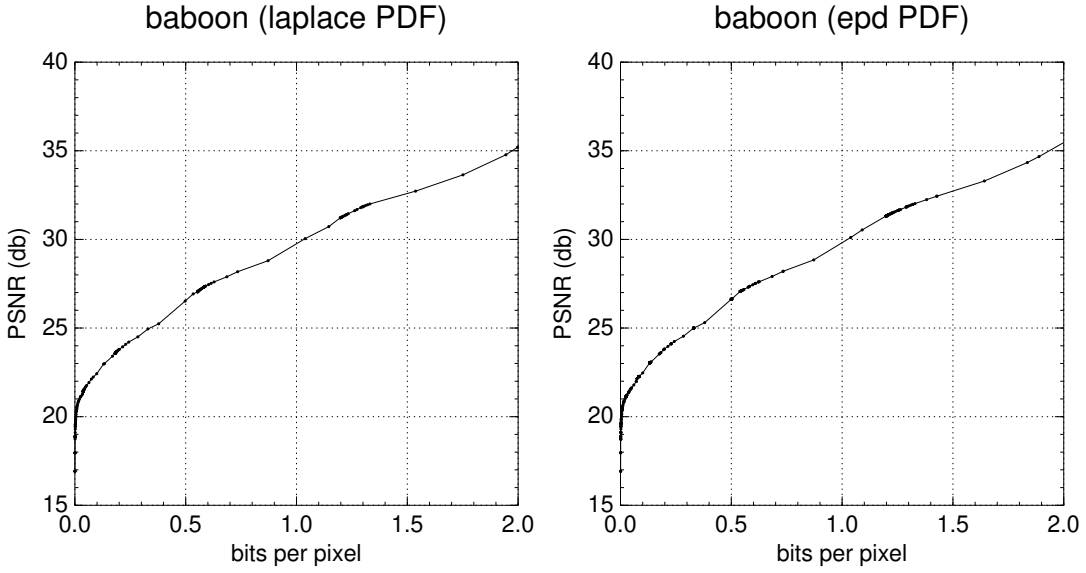


FIGURE 7.7: Baboon rate-distortion curve ($\bar{s} = 0.7$, $\bar{\sigma} = 17.4481$)

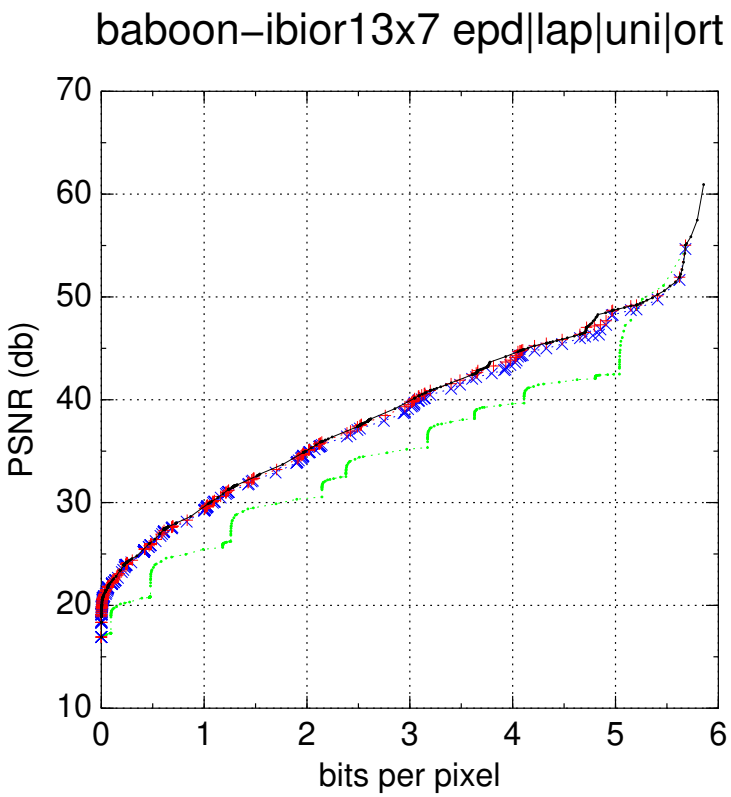


FIGURE 7.8: Baboon rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

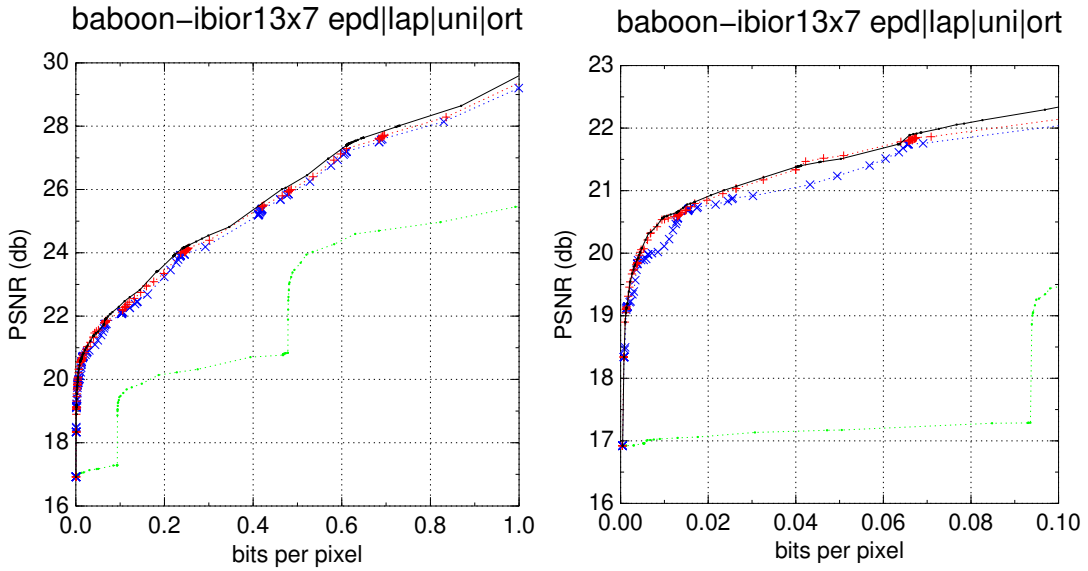


FIGURE 7.9: Low bit rate zoom for the baboon rate-distortion curve

7.2.3 Barbara

FIGURE 7.10: Barbara

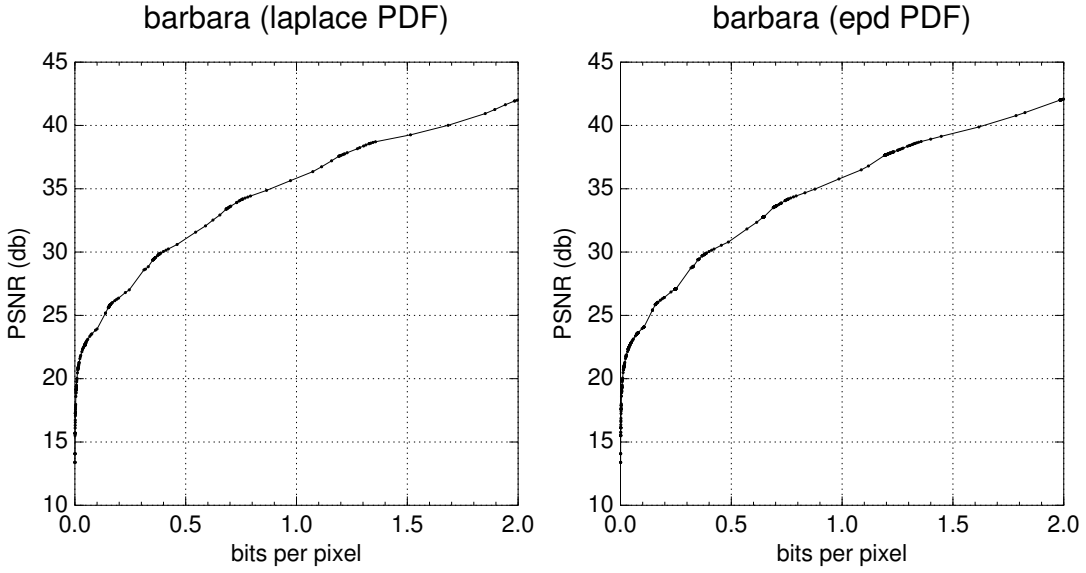


FIGURE 7.11: Barbara rate-distortion curve ($\bar{s} = 0.45$, $\bar{\sigma} = 16$)

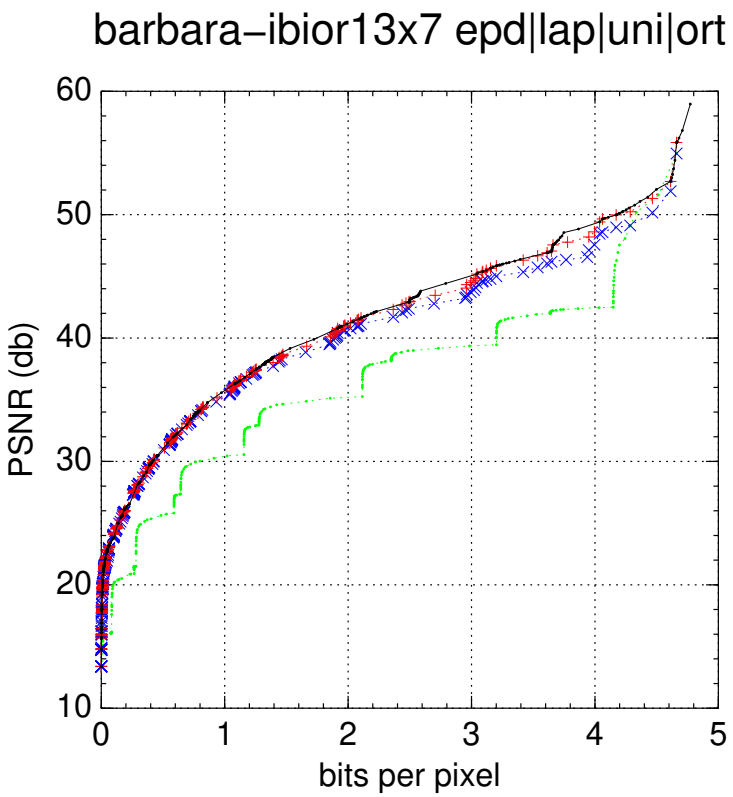


FIGURE 7.12: Barbara rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

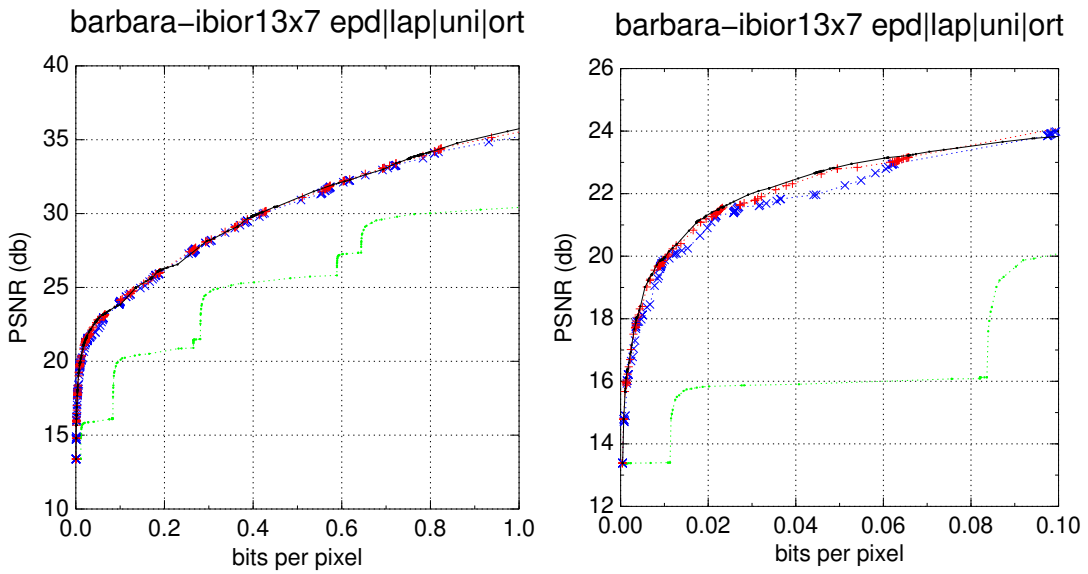


FIGURE 7.13: Low bit rate zoom for the barbara rate-distortion curve

7.2.4 Lena

FIGURE 7.14: Lena

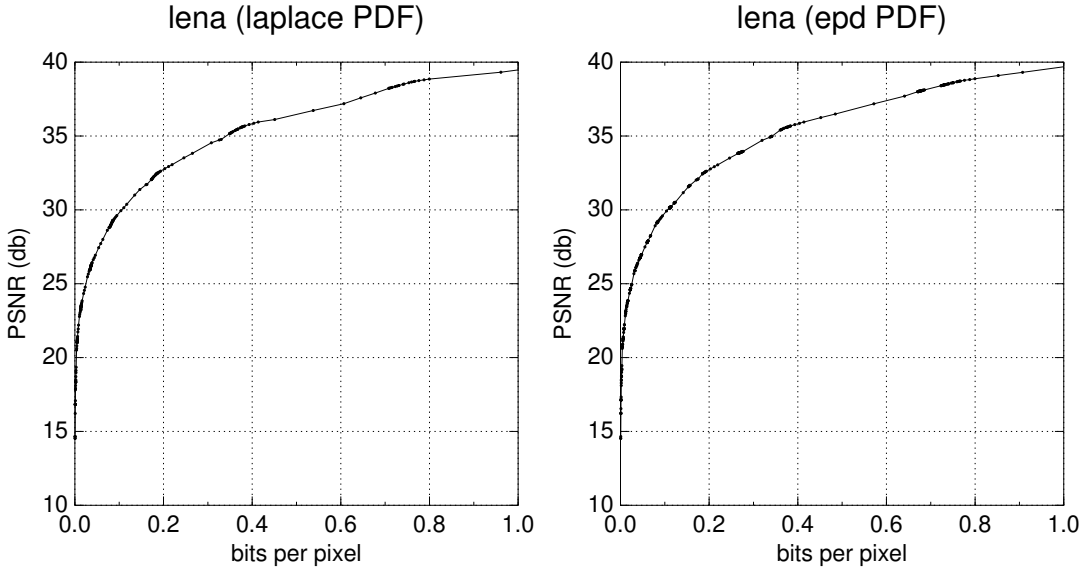


FIGURE 7.15: Lena rate-distortion curve ($\bar{s} = 0.45$, $\bar{\sigma} = 7.02501$)

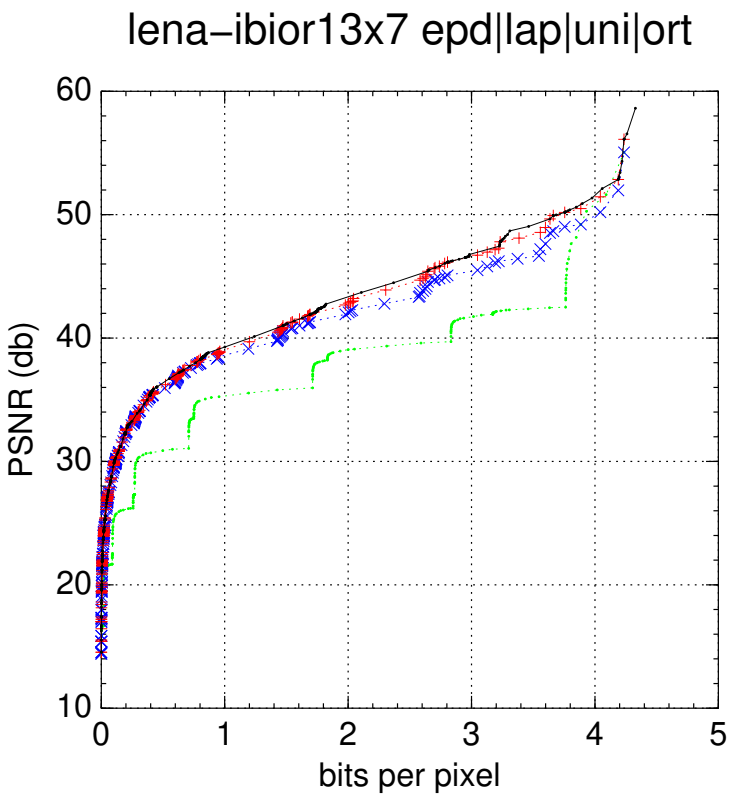


FIGURE 7.16: Lena rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

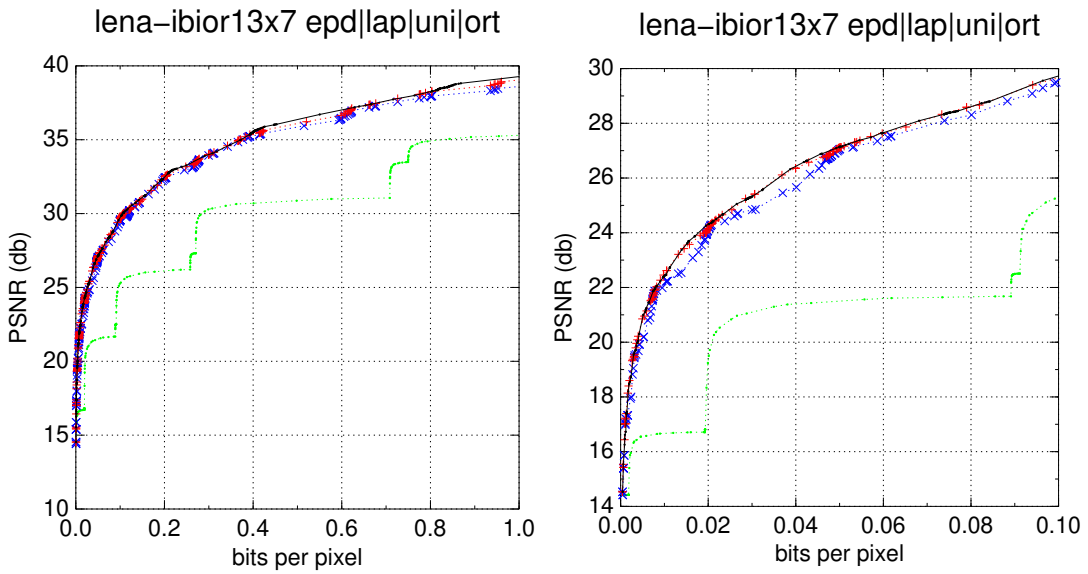


FIGURE 7.17: Low bit rate zoom for the lena rate-distortion curve

7.2.5 Peppers

FIGURE 7.18: Peppers

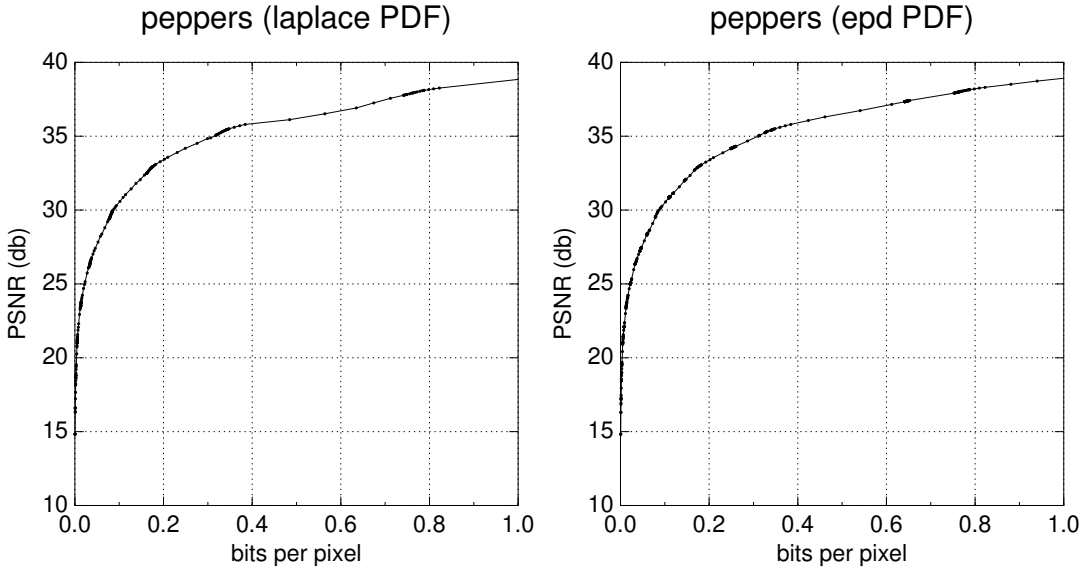


FIGURE 7.19: Peppers rate-distortion curve ($\bar{s} = 0.45$, $\bar{\sigma} = 7.33603$)

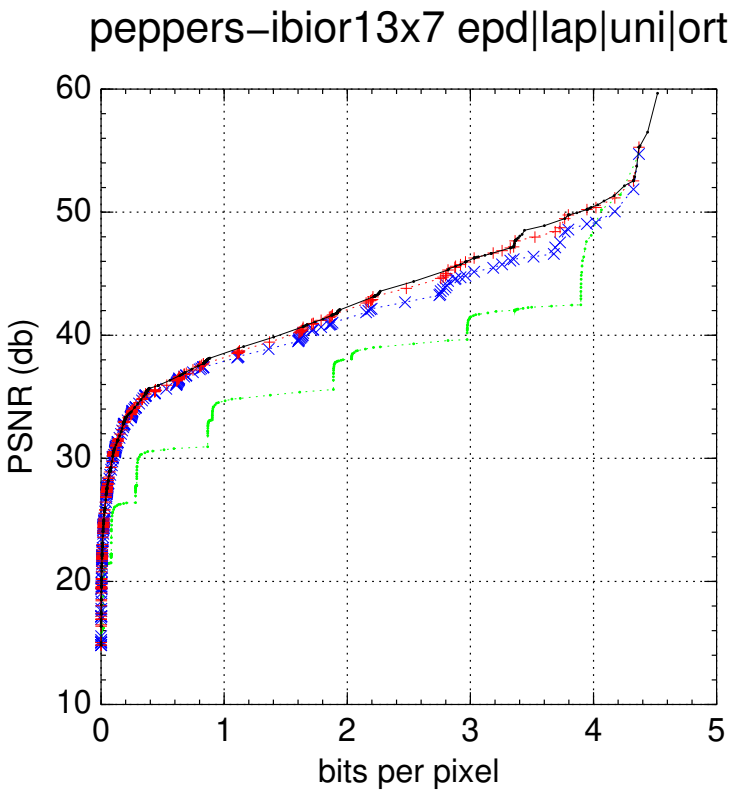


FIGURE 7.20: Peppers rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

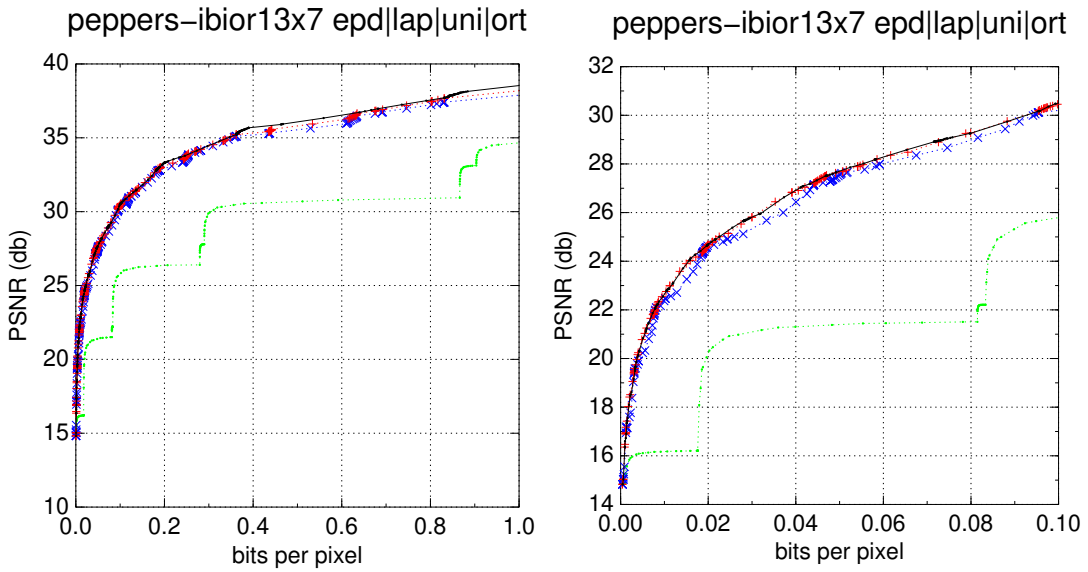


FIGURE 7.21: Low bit rate zoom for the peppers rate-distortion curve

7.2.6 Sailboat

FIGURE 7.22: Sailboat

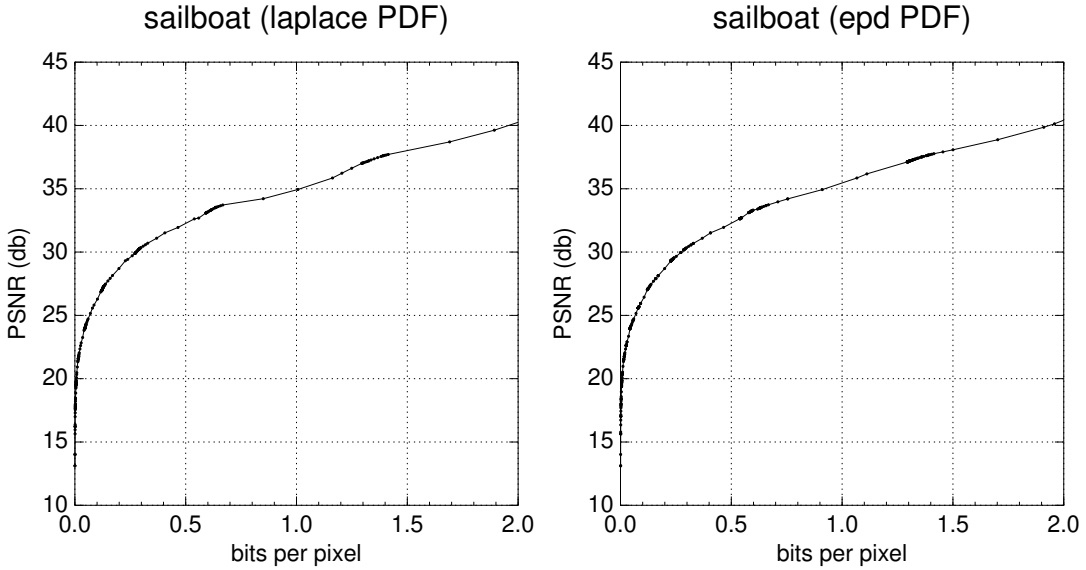


FIGURE 7.23: Sailboat rate-distortion curve ($\bar{s} = 0.6$, $\bar{\sigma} = 9.93486$)

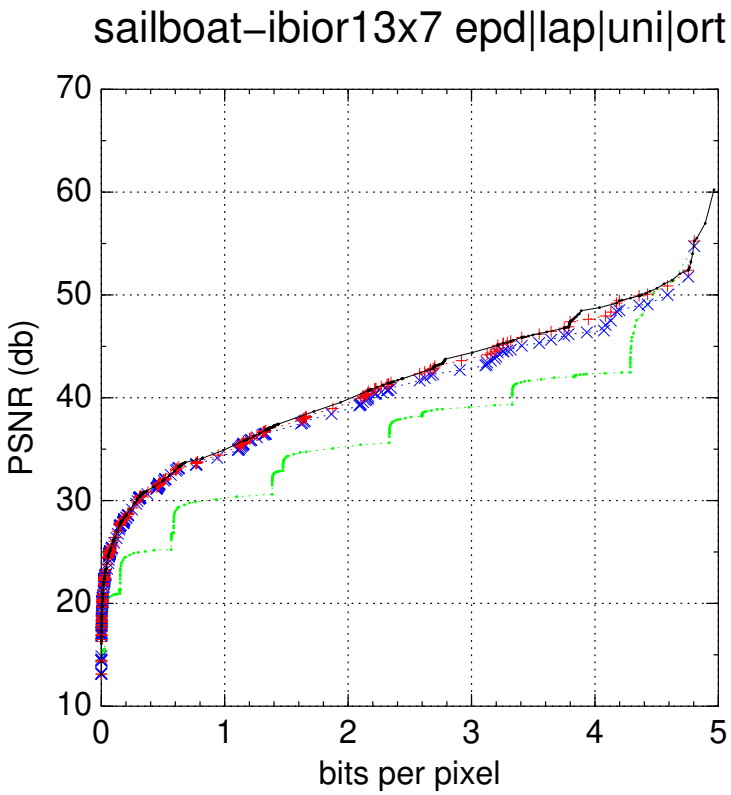


FIGURE 7.24: Sailboat rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

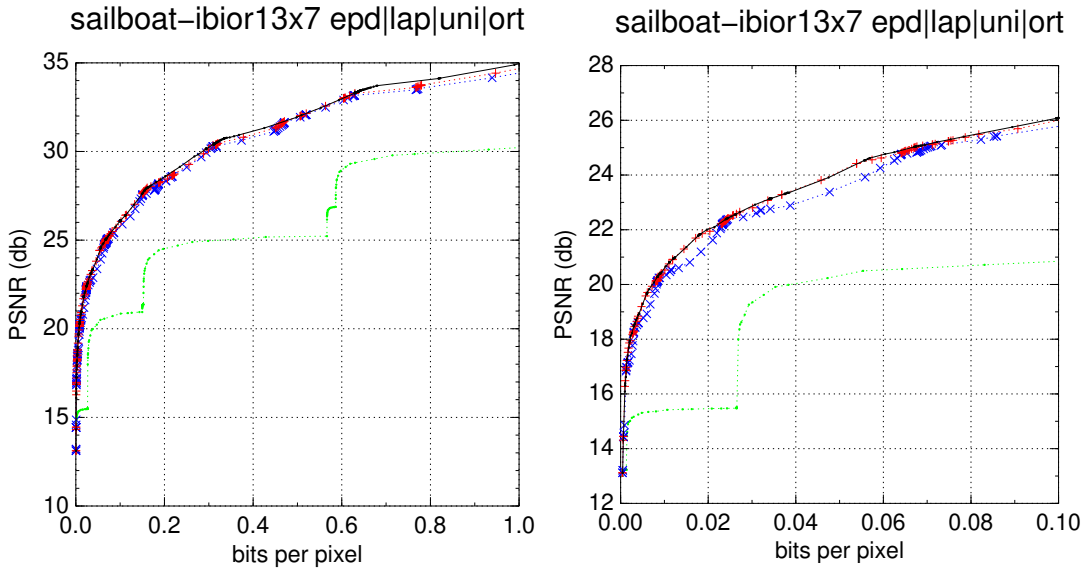


FIGURE 7.25: Low bit rate zoom for the sailboat rate-distortion curve

7.2.7 DEBT × JPEG-2000 Compression Comparison

Table 7.7: Airplane PSNR

rate	size (bytes)	jp2k (db)	debt (db)	diff (db)
0.001	260	20.30	21.87	1.57
0.002	520	22.48	23.41	0.93
0.003	778	23.76	24.34	0.58
0.004	1037	24.69	25.21	0.52
0.005	1308	25.54	25.71	0.17
0.006	1526	26.00	26.07	0.07
0.007	1827	26.60	26.60	0.00
0.008	2087	27.18	26.94	-0.24
0.009	2351	27.62	27.36	-0.26
0.010	2476	27.83	27.58	-0.25

Table 7.8: Baboon PSNR

rate	size (bytes)	jp2k (db)	debt (db)	diff (db)
0.001	255	19.75	20.43	0.68
0.002	493	20.57	20.84	0.27
0.003	783	20.84	21.10	0.26
0.004	1026	21.06	21.26	0.20
0.005	1283	21.29	21.43	0.14
0.006	1384	21.36	21.49	0.13
0.007	1718	21.57	21.67	0.10
0.008	2065	21.74	21.85	0.11
0.009	2226	21.82	21.94	0.12
0.010	2610	22.00	22.22	0.22

Table 7.9: Barbara PSNR

rate	size (bytes)	jp2k (db)	debt (db)	diff (db)
0.001	252	17.89	19.67	1.78
0.002	520	20.22	20.95	0.73
0.003	770	21.17	21.67	0.50
0.004	998	21.64	22.07	0.43
0.005	1267	22.05	22.53	0.48
0.006	1571	22.38	22.85	0.47
0.007	1675	22.52	22.94	0.42
0.008	2080	22.92	23.28	0.36
0.009	2270	23.18	23.44	0.26
0.010	2572	23.46	23.61	0.15

Table 7.10: Lena PSNR

rate	size (bytes)	jp2k (db)	debt (db)	diff (db)
0.001	258	20.27	21.96	1.69
0.002	504	22.70	23.69	0.99
0.003	729	23.87	24.62	0.75
0.004	1043	25.14	25.83	0.69
0.005	1309	26.00	26.50	0.50
0.006	1561	26.65	26.97	0.32
0.007	1815	27.15	27.50	0.35
0.008	2081	27.55	27.99	0.44
0.009	2342	28.03	28.48	0.45
0.010	2602	28.45	28.98	0.53

Table 7.11: Peppers PSNR

rate	size (bytes)	jp2k (db)	debt (db)	diff (db)
0.001	247	19.47	21.94	2.47
0.002	523	23.11	24.06	0.95
0.003	762	24.52	25.11	0.59
0.004	1041	25.72	26.21	0.49
0.005	1205	26.26	26.66	0.40
0.006	1565	27.44	27.52	0.08
0.007	1808	28.04	28.04	0.00
0.008	2082	28.54	28.54	0.00
0.009	2316	28.98	28.97	-0.01
0.010	2604	29.52	29.57	0.05

Table 7.12: Sailboat PSNR

rate	size (bytes)	jp2k (db)	debt (db)	diff (db)
0.001	247	18.68	20.18	1.50
0.002	511	20.79	21.60	0.81
0.003	781	21.82	22.42	0.60
0.004	990	22.52	22.93	0.41
0.005	1300	23.18	23.69	0.51
0.006	1533	23.67	24.17	0.50
0.007	1682	23.93	24.36	0.43
0.008	2065	24.64	24.87	0.23
0.009	2352	25.05	25.26	0.21
0.010	2616	25.34	25.59	0.25

7.3 High Resolution Images

Even though the classic images shown previously are essential for comparison purposes, they are considered small by today's standards. A set of higher resolution, high quality photos photos are available for the purpose of image compression comparison at http://imagecompression.info/test_images/gray8bit.zip and are much better suited for comparing the results of using an EPD PDF instead of a predefined laplace PDF due to the increased statistical significance of using a higher number of samples (coefficients).

The images used for comparison were "Bridge" (2749 × 4049), "Cathedral" (2000 × 3008), "Moon" (1986 × 1986), and "Tree" (6088 × 4550).

The image "Moon" is available as a 24-bit color image at http://www.hlevkin.com/TestImages/testpat.bmp which is then processed by the following command "`bmptoppm < moon.bmp | ppmtopgm > moon.pgm`" in order to obtain the final gray scale image used to plot the rate distortion curves.

It can be observed that the use of an EPD PDF results in much smoother rate distortion curves by simply changing the order of the significance and refinement information as described in chapter 4. The DEBT algorithm was, just like with the standard images, used with the CDF-9/7 DWT with $\langle 1, 2 \rangle$ normalization, 6 levels of decomposition, and only blocks.

As it was done for the classic images, the rate distortion curves are also compared using the integer ibior13x7 DWT for all images.

Many other images were compared (which are not shown) and all present similar results, depending on how close the measured shape parameter s is to a laplace PDF ($s = 1$), i.e., the farther from $s = 1$ the more pronounced were the improvements, as expected.

7.3.1 Bridge

FIGURE 7.26: Bridge

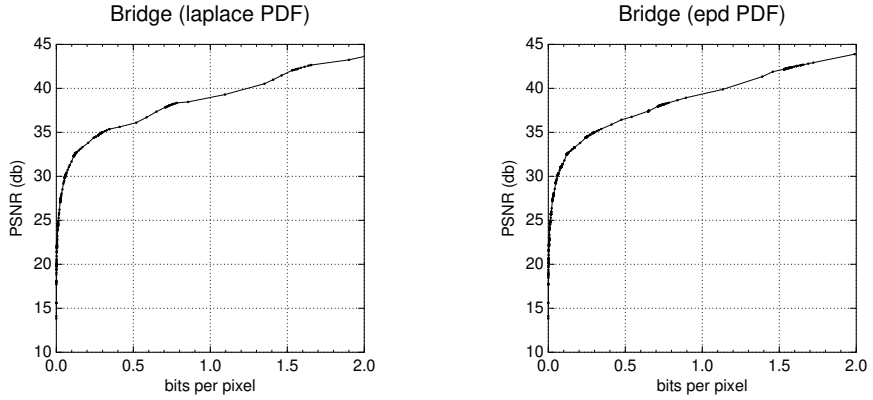


FIGURE 7.27: Bridge rate-distortion curve ($\bar{s} = 0.45$, $\bar{\sigma} = 5.41702$)

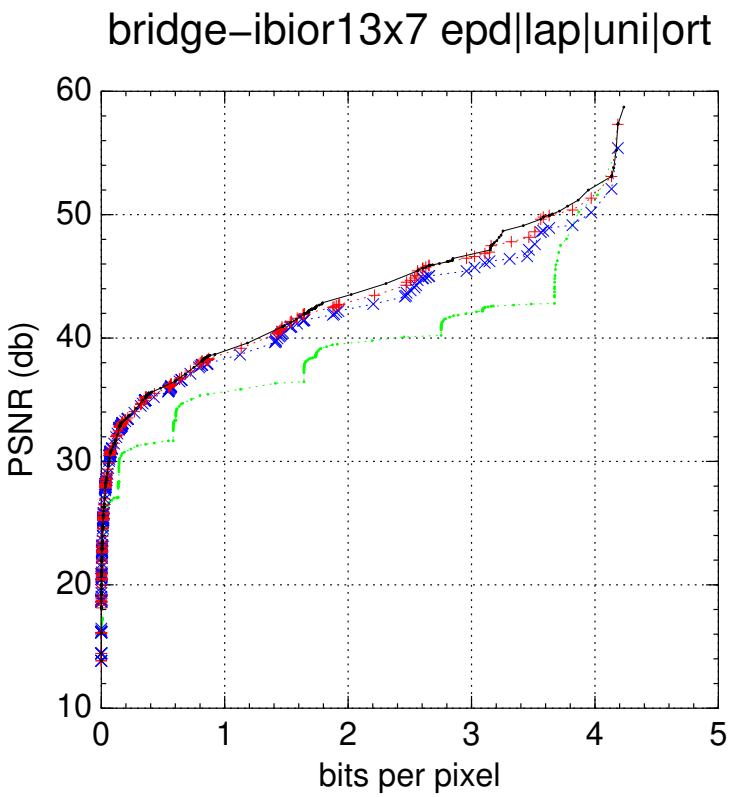


FIGURE 7.28: Bridge rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

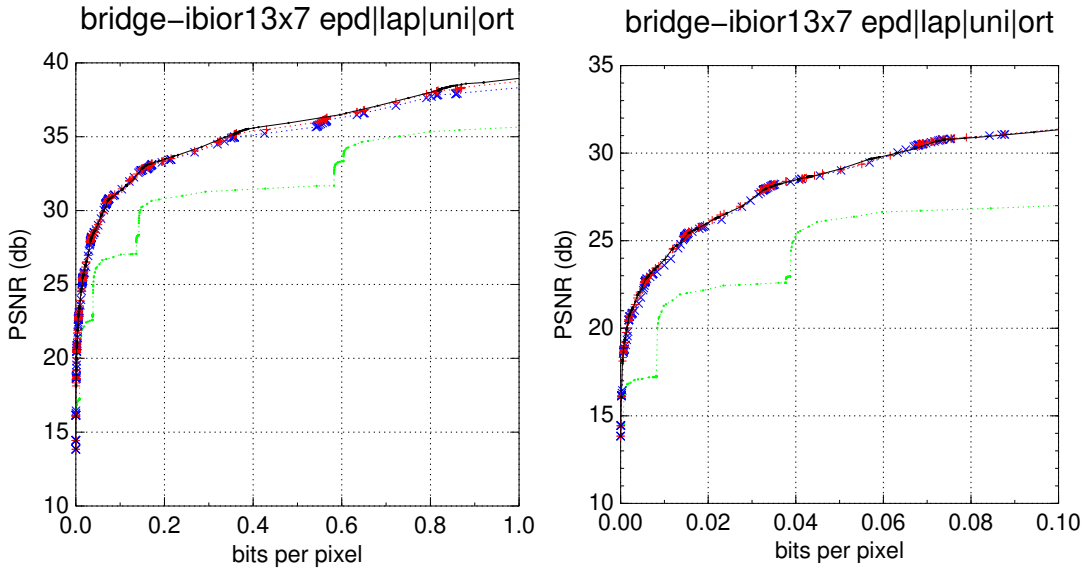


FIGURE 7.29: Low bit rate zoom for the bridge rate-distortion curve

7.3.2 Cathedral

FIGURE 7.30: Cathedral

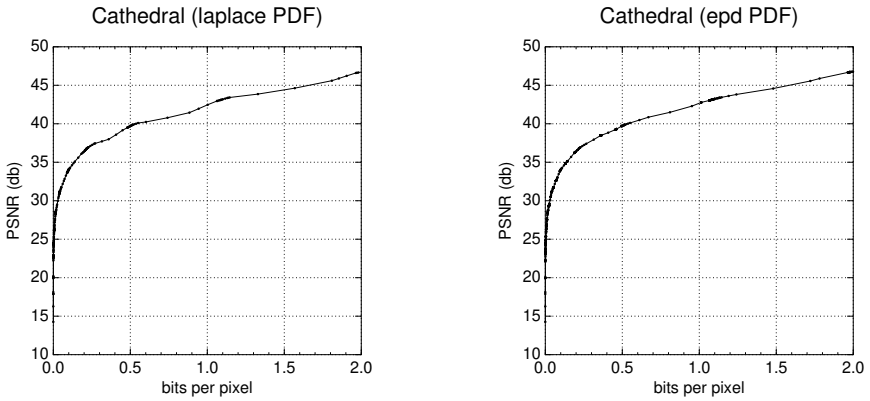


FIGURE 7.31: Cathedral rate-distortion curve ($\bar{s} = 0.35$, $\bar{\sigma} = 4.36203$)

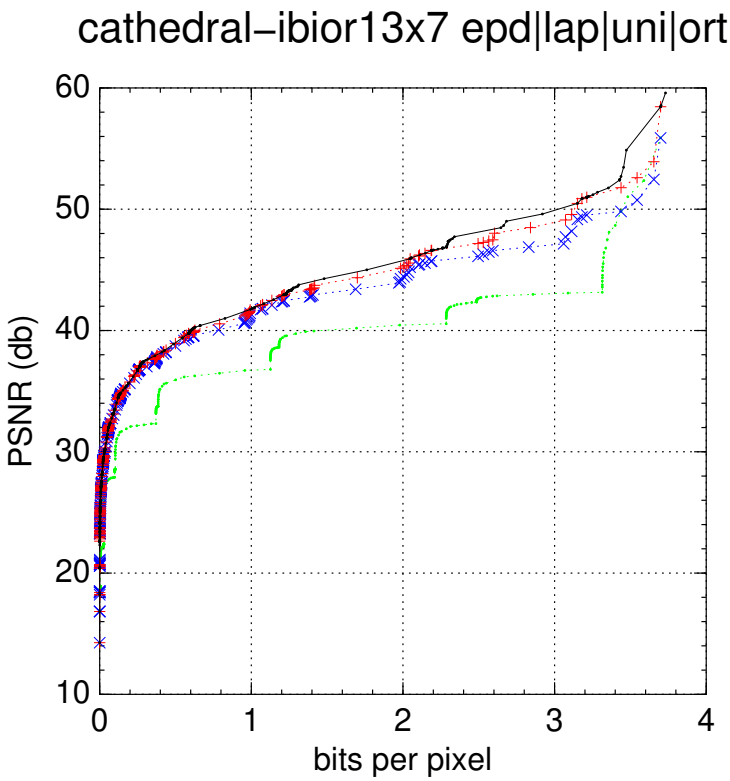


FIGURE 7.32: Cathedral rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

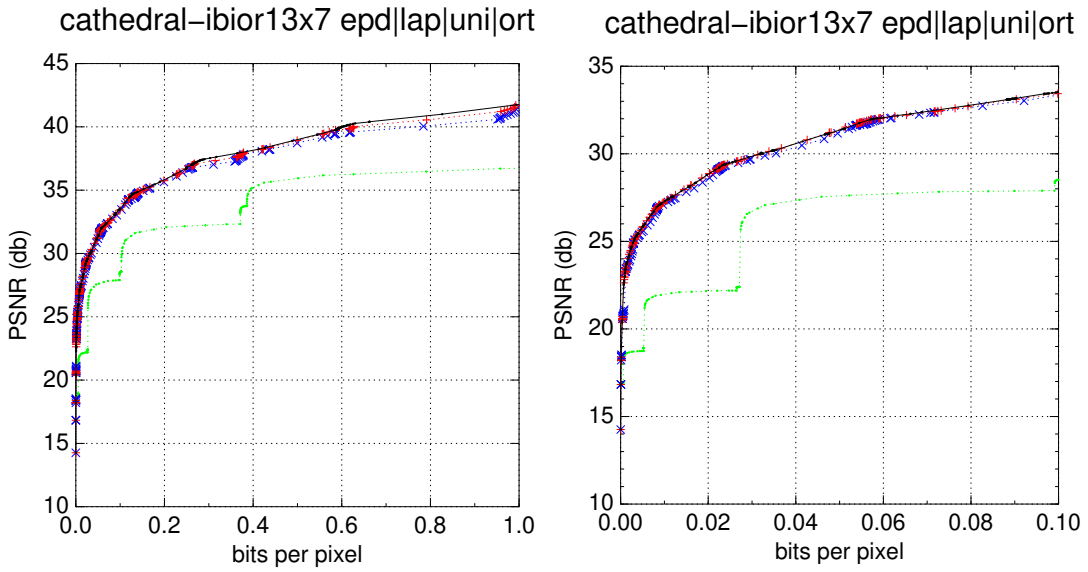


FIGURE 7.33: Low bit rate zoom for the cathedral rate-distortion curve

7.3.3 Moon

FIGURE 7.34: Moon

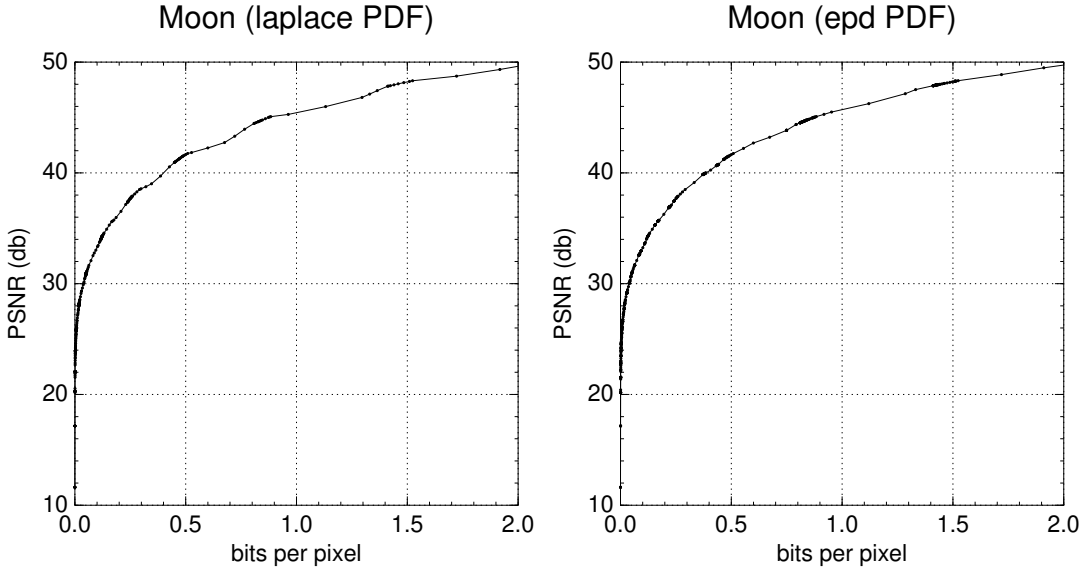


FIGURE 7.35: Moon rate-distortion curve ($\bar{s} = 0.25$, $\bar{\sigma} = 5.65685$)

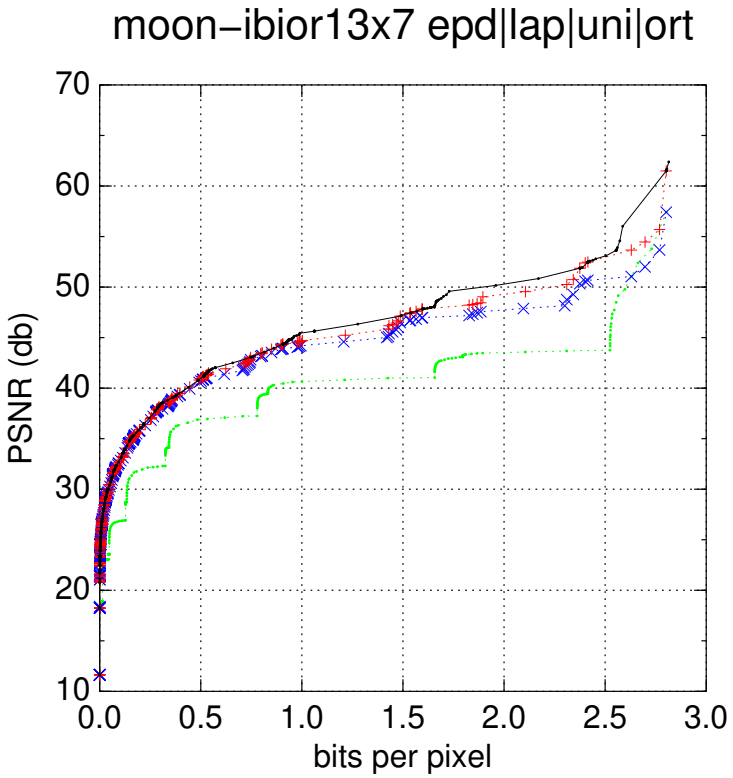


FIGURE 7.36: Moon rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

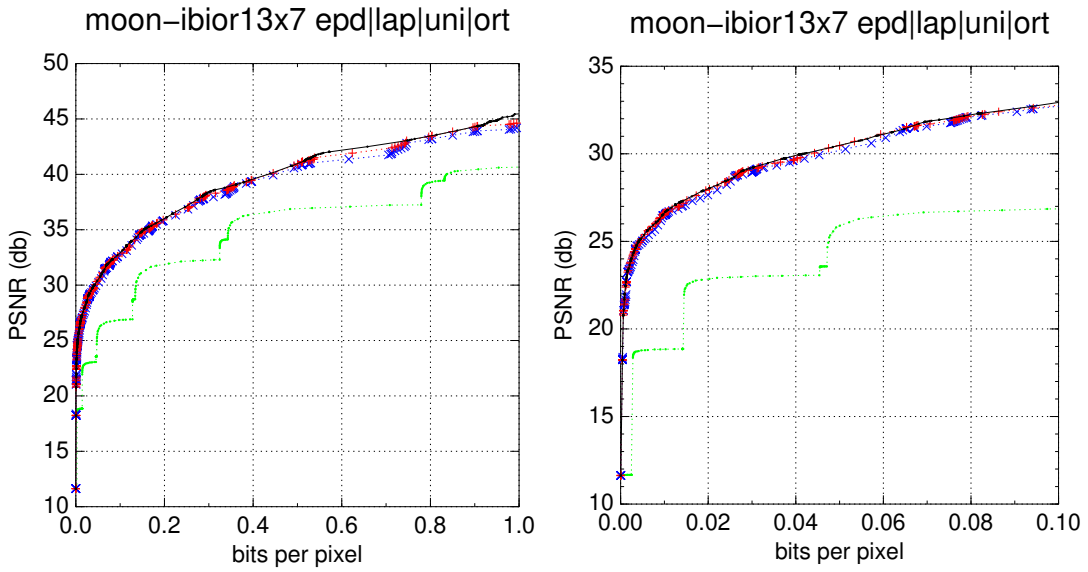


FIGURE 7.37: Low bit rate zoom for the moon rate-distortion curve

7.3.4 Tree

FIGURE 7.38: Tree

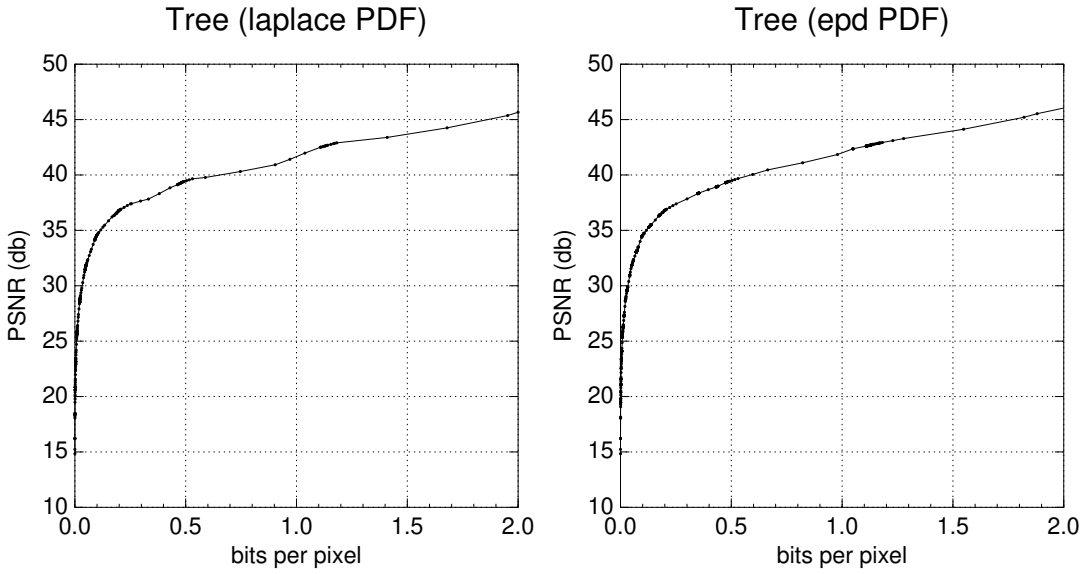


FIGURE 7.39: Tree rate-distortion curve ($\bar{s} = 0.4$, $\bar{\sigma} = 4.36203$)

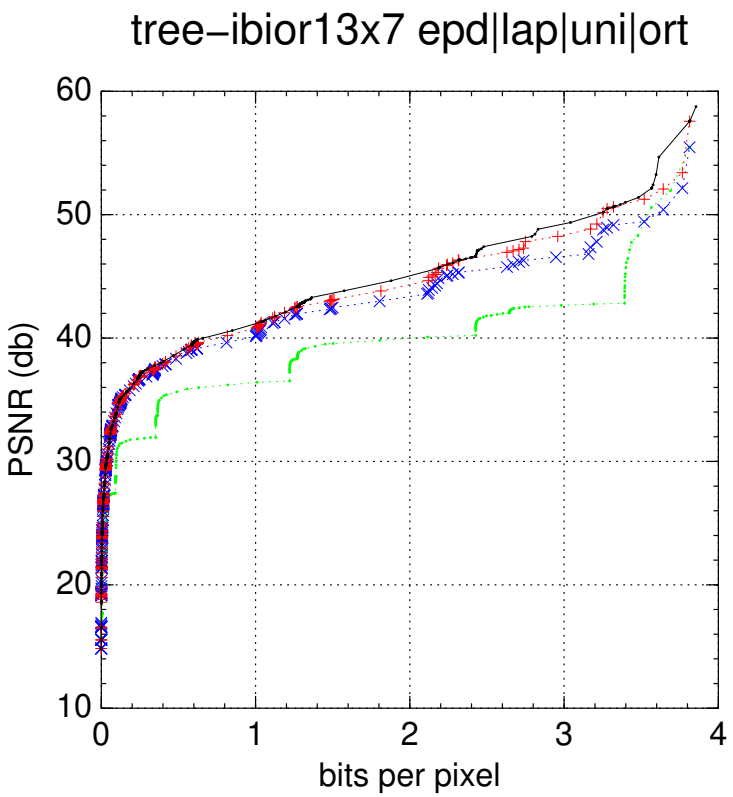


FIGURE 7.40: Tree rate-distortion curve using the ibior13x7 DWT for the EPD (black), Laplace (red), Uniform (blue), and Uniform with orthogonal ordering (green)

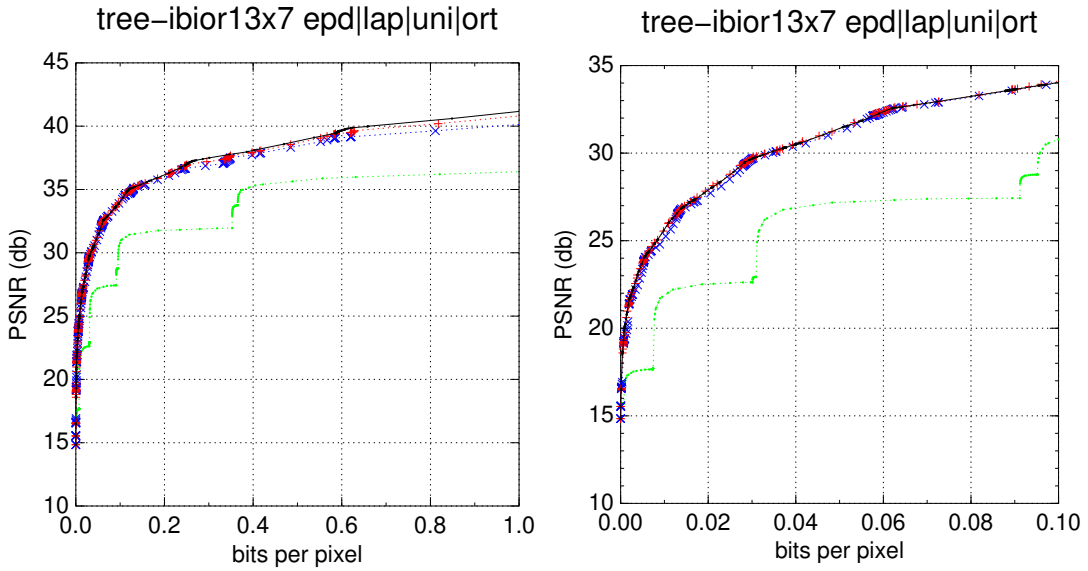


FIGURE 7.41: Low bit rate zoom for the tree rate-distortion curve

7.4 Underwater Images

In order to test and tune some parameters of the DEBT algorithm to underwater imagery, we used a set of 1258 underwater gray-scale images from the Australian Center for Field Robotics (http://marine.acfr.usyd.edu.au/datasets/data/TasCPC/TasCPC_LC.tar.gz) as input for a low bitrate compression performance comparison against the JPEG-2000 [23] algorithm using the same parameters used for the classical images.

7.4.1 EPD Shape Parameter

All images were investigated in order to find out the shape parameter of the EPD PDF for each image. Figure 7.42 shows the resulting shape parameters by using the variance (vshape) and the kurtosis (kshape) methods using the CDF-9/7 DWT with $\langle 1, 2\rangle$ normalization and 6 decompositon levels. The average value of vshape was 0.79969 and the average value of kshape was 0.79509, which are quite close and point in the direction of a good fit of the data to an EPD PDF and are close to the shape parameter of a laplace PDF, which has shape parameter $s = 1$.

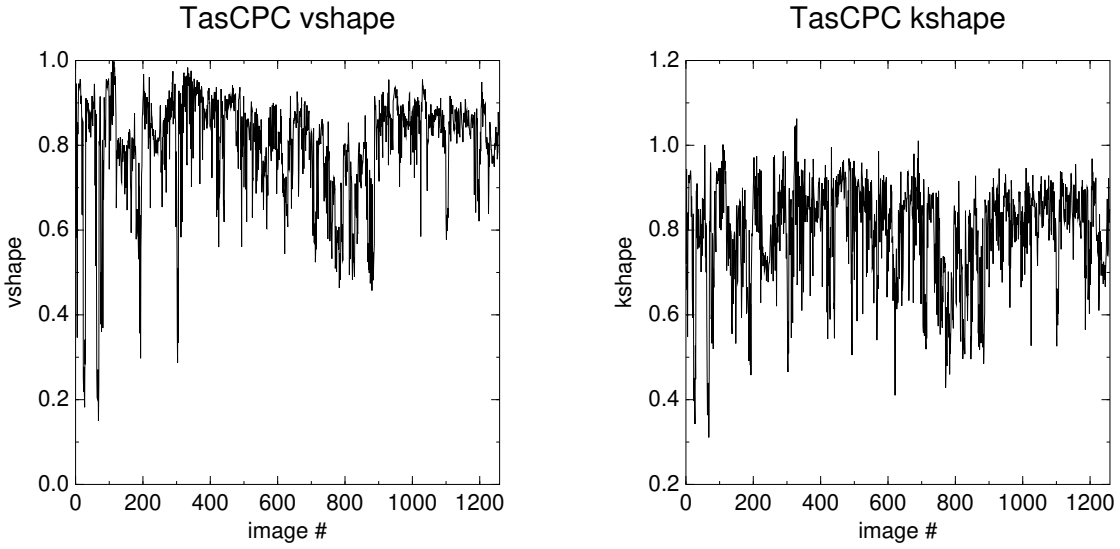


FIGURE 7.42: TasCPC shape (from variance and kurtosis)

This means that DEBT could be used with a predefined laplace PDF instead of calculating the shape parameters and standard deviation of a fitted EPD PDF without much difference on average for this class of images.

7.4.2 DEBT × JPEG-2000 Compression Comparison

The PSNR difference between DEBT and JPEG-2000 [23] algorithms was plotted for 6 different rates: 1:10000 (0.0001), 1:5000 (0.0002), 1:2000 (0.0005), 1:1000 (0.001), 1:200 (0.005), and 1:100 (0.01). All images are 1360×1024 pixels grayscale and were numbered in lexical order from 1 up to 1258.

For the first 2 rates, i.e., 1:10000 (0.0001) and 1:5000 (0.0002), the JPEG-2000 [23] implementation "`JasPer`" [78] generated a "`warning: empty layer generated`" message for all images at the 1:10000 (0.00001) rate and a compressed file with 138 bytes was generated in all cases. For the 1:5000 (0.0002) ratio, this same message was issued for 422 images which, in this case, resulted in the same 138 byte long compressed file. This may explain the huge difference in favor of DEBT for these very high compression ratios, i.e., these compression ratios are too high for either the JPEG-2000 [23] algorithm itself or its implementation "`JasPer`" [78].

Figure 7.43 shows the PSNR compression difference in db for all images for rates 0.0001 and 0.0002 where it can be observed that the JPEG-2000 [23] implementation used could not cope with such high compression ratios. Nevertheless, as DEBT is an bit oriented progressive algorithm, all bits following the header contribute to reduce distortion and some useful images are generated for these tiny sizes.

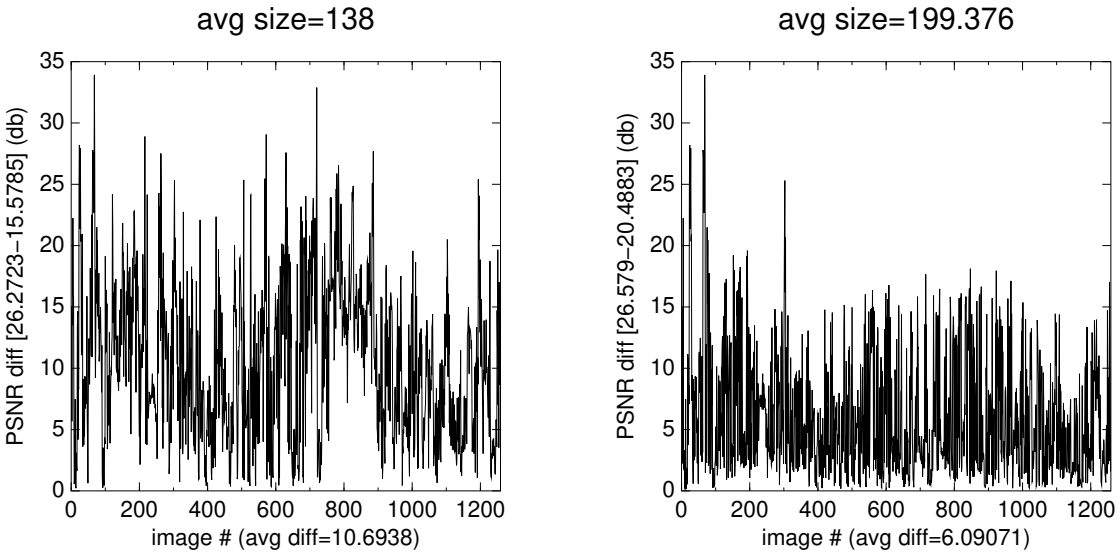


FIGURE 7.43: TasCPC images for rates 0.0001 and 0.0002

Fig. 7.44 shows the PSNR compression difference in db for all images for rates 0.0005 and 0.001. It is quite clear that the DEBT algorithm performs better for all images, without

exception, for these lower rates by quite a significant margin, reaching a difference of 2.68 db in the 0.0005 case and 2.20 db in the 0.001 case. On average, the gain of DEBT over JPEG-2000 [23] is 0.45 db and 0.31 db for rates 0.0005 and 0.001, respectively.

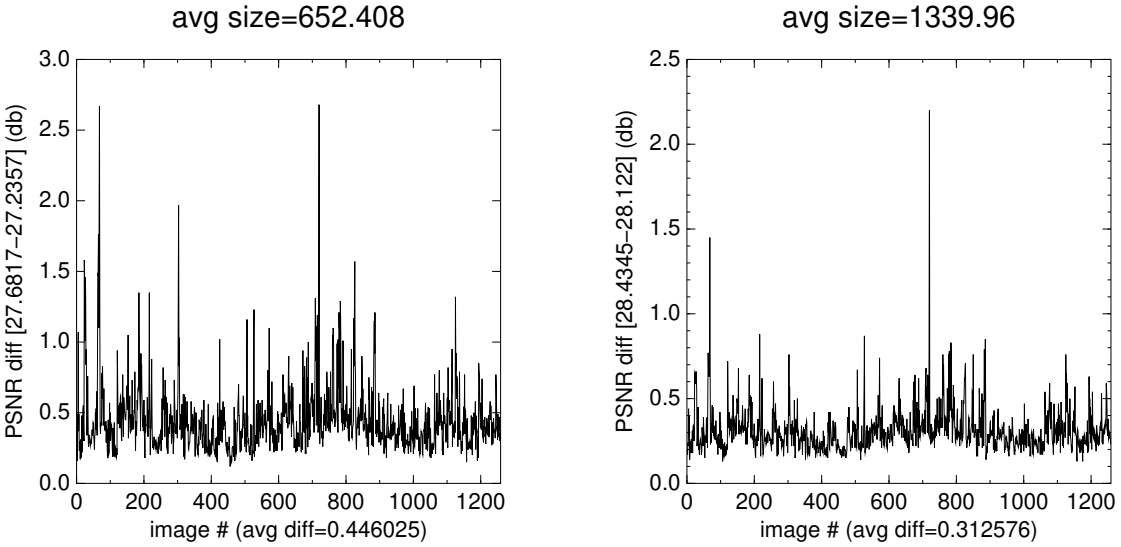


FIGURE 7.44: TasCPC images for rates 0.0005 and 0.001

As the rate goes up (less compression and more quality), the DEBT algorithm is still superior to the JPEG-2000 [23] for all 1258 images in this dataset except for 3 in the 0.005 case and 4 in the 0.01 case and in all these cases the compression quality was very high (the images were very dark), over 40 db or higher for both algorithms. Fig. 7.45 shows the results for these rates. On average, the gain of DEBT over JPEG-2000 [23] is 0.25 db and 0.27 db for rates 0.005 and 0.01, respectively.

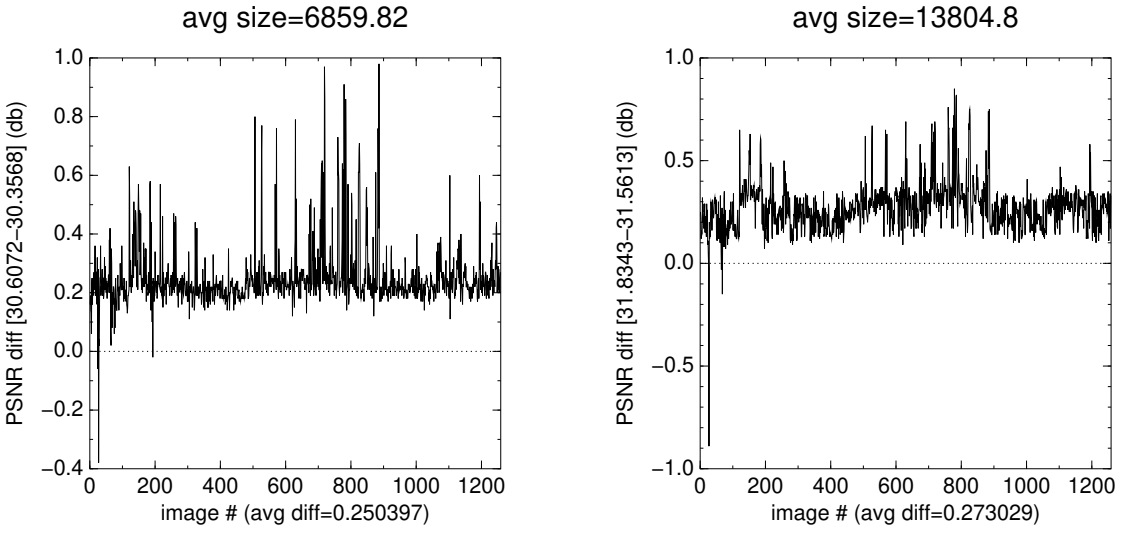


FIGURE 7.45: TasCPC images for rates 0.005 and 0.01

A few images were randomly selected from this dataset and are presented in figure 7.46. Each row shows the original image (1st column) and compressed with the DEBT algorithm at exactly 500, 1000, and 2000 bytes on the 2nd, 3rd, and 4th columns, respectively.

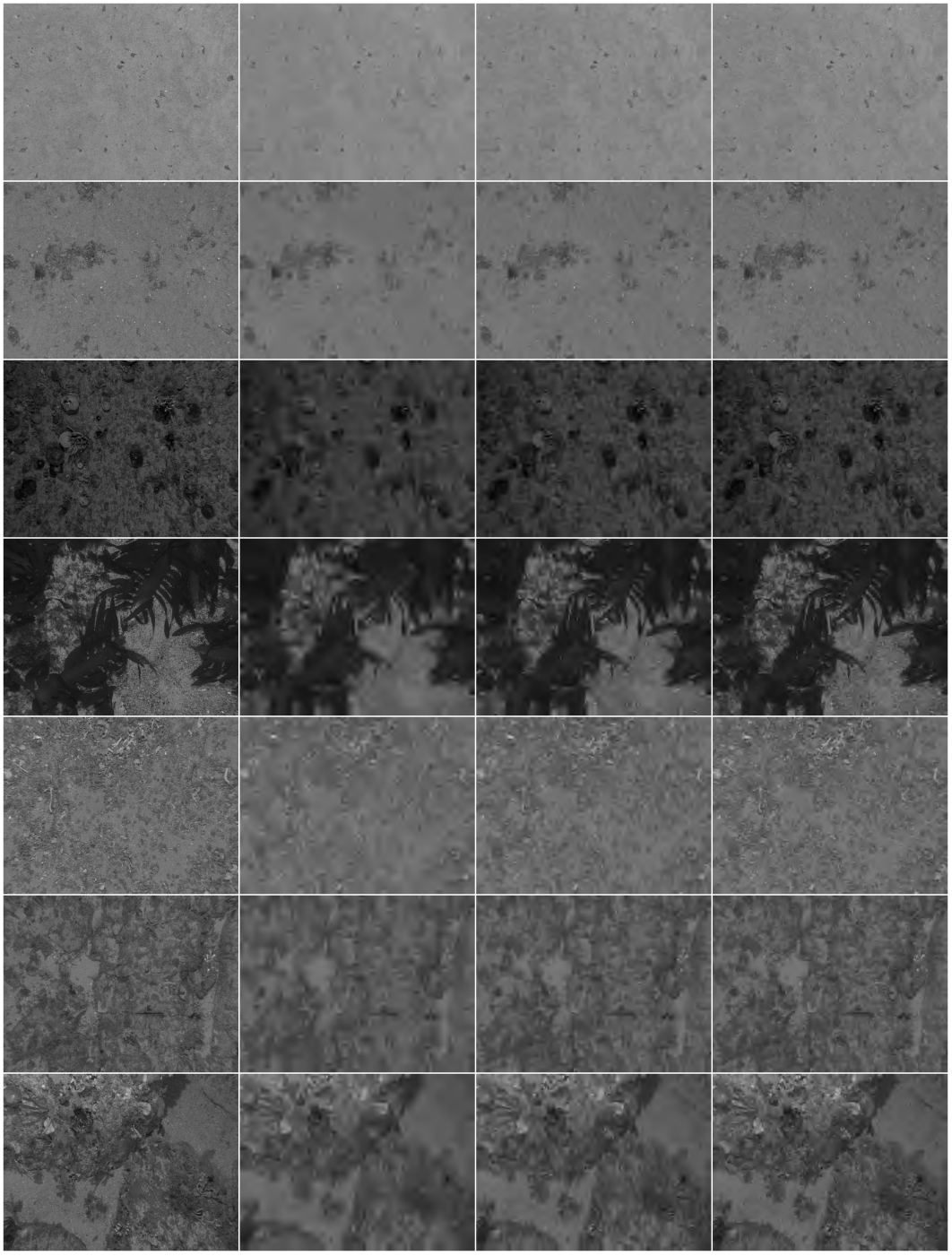


FIGURE 7.46: Underwater images 16, 227, 274, 813, 977, 1082, and 1219 compressed at original, 500, 1000, and 2000 bytes

An important point to note is that the DEBT algorithm makes it very easy, by simply changing the priority list order, to create a stream which is not optimal in the MSE sense but which could be better suited to highlight other characteristics of the image at very low bitrates. These alternate metrics should be the subject of further investigation.

7.4.3 DEBT × JPEG-2000 Timing Comparison

It should be taken into consideration that software running times serve only as an indication of rough complexity since many implementation optimizations are possible and it is difficult to compare if all tested implementations have been optimized. However, timing measurements do provide a useful indication of relative practical complexity.

A run for the rate 0.001 was timed using both DEBT and JPEG-2000 [23] using an Intel Desktop computer based on a Core-i7 2600k CPU running at 3.4GHz. Also, the DEBT algorithm and its transform were run single threaded, i.e., with the environment variable "`OMP_NUM_THREADS=1`". Because we need to find out the exact compressed size of each resulting JPEG-2000 [23] compressed file and use it as the maximum compressed size for DEBT, we ran the compression loop 2 times, once with JPEG-2000 [23] followed by DEBT, and another consisting of running only the JPEG-2000 [23] compressor. The difference between the first and the second should give an idea of the DEBT running time. Each compression generated a file in the file system for both algorithms so that both input and output the exact same amount of data. The results of timing the script that ran the algorithms as described for the 1258 images of this set are given in Table 7.13. In this table we have also included a run of the DEBT algorithm using the ibior13x7 integer DWT (iDEBT) and the timings obtained with this transform are given in the last column.

TABLE 7.13: DEBT and JPEG-2000 timings (1258 images) - ratio 0.001

	JPEG-2000 & DEBT	JPEG-2000	JPEG-2000 & iDEBT
real	4m28.038s	4m12.720s	4m23.429
user	4m14.317s	4m04.805s	4m11.487
sys	0m14.071s	0m08.056s	0m12.303s

Assuming that all I/O is included in the line corresponding the system calls "sys", including executable load and link times, we are mainly interested in the "user" line. It can be seen that the time for both algorithms is 254.317 sec while the time taken for

the JPEG-2000 [23] alone is 244.805 sec. This means that the time taken for DEBT is 9.512 sec which corresponds to approximately 132.25 frames per second while the JPEG-2000 [23] rate is about 5.14 frames per second. This means that, in this case, our DEBT implementation is about 25.73 times faster than the "`JasPer`" [78] JPEG-2000 [23] implementation. If we use the timings for the DEBT algorithm using the ibior13x7 integer DWT, the measured DEBT running time is 6.682 sec which gives us 188.27 frames per second and a speedup of approximately 36.63 times over "`JasPer`" [78].

7.4.4 DEBT Lossless Timing and Compression Comparison

We have also run a test comparing DEBT lossless performance against JPEG-LS [39] lossless compressor, JPEG-2000 [23] in lossless mode using the 5/3 integer DWT, and PNG, a popular file format in which lossless compression is achieved through prediction in two passes, in order to find out how well DEBT performs in the lossless case, even though neither JPEG-LS [39] nor PNG are progressive encoders and the decoder must have the whole compressed stream in order to decode it. Once again, DEBT can be truncated at any point for the transmission of an approximation of the image while the lossless file can be stored for later retrieval. Also, the DEBT algorithm was run in single-threaded mode by setting the environment variable "`OMP_NUM_THREADS=1`".

Table 7.14 shows the timings for running DEBT with the ibior13x7 DWT on the first column, JPEG-LS [39] ("`locoe`") on the second column, DEBT with the ibior5x3 DWT on the third column, JPEG-2000 [23] ("`jasper`") in the fourth column, and PNG ("`pamtopng`") in the fifth column. The final average size (in bytes) of the lossless files for the 1258 images are displayed on the last line for each algorithm.

TABLE 7.14: DEBT lossless timings (1258 images)

	DEBT 13/7	JPEG-LS	DEBT 5/3	JPEG-2000	PNG
real	0m59.185s	1m28.843s	0m59.047s	4m22.583s	2m56.036s
user	0m52.861s	1m26.870s	0m52.930s	4m13.471s	2m53.898s
sys	0m06.580s	0m02.182s	0m06.370s	0m09.308s	0m02.287s
avg. size	777054	741408	782720	756983	762291

Once again, using the "user" line as the effective timings, it can be seen that DEBT is the fastest algorithm while JPEG-LS is the one that compresses the best. DEBT took approximately 53 seconds while JPEG-LS took approximately 1 minute and 27

seconds, JPEG-2000 [23] took approximately 4 minutes and 13 seconds, and PNG took approximately 2 minutes and 54 seconds to compress all 1258 images. More exactly, they process frames at an average rate of 23.80, 14.48, 4.96, and 7.23 frames per second for DEBT-ibior13x7, JPEG-LS, JPEG-2000 [23], and PNG, respectively. DEBT-ibior5x3 has approximately the same performance as DEBT-ibior13x7 within a margin of error. On average, DEBT is 64% (1.64) faster than JPEG-LS, 380% (4.80) faster than JPEG-2000 [23], and 229% (3.29) faster than PNG for this set of images.

The average compressed size of JPEG-LS was the smallest, with an average of 741408 bytes, followed by JPEG-2000 [23] with 756983 bytes (2.1% higher), followed by PNG with 762291 bytes (2.8% higher) followed by DEBT with the ibior13x7 DWT with 777054 bytes (4.8% higher), followed by DEBT with the ibior5x3 DWT with 782720 bytes (5.6% higher).

In summary, the DEBT algorithm compresses at 64% higher speed than JPEG-LS while producing 5% larger files, or at 380% higher speed than JPEG-2000 [23] while producing 2.7% larger files, or at 229% higher speed than PNG while producing 1.9% larger files. Also, the files produced by DEBT are bit-oriented progressive streams and can be simply truncated and have its prefix transmitted so that the same file can be used for both local archival and transmission.

The total lack of an entropy coding step and the fact that the inter band correlation is not being used at all (all runs were based on using blocks only) coupled with an optimized DWT implementation makes DEBT very fast but produce slightly larger lossless compressed files.

7.5 Region Of Interest

In a low-bandwidth scenario or when the images are highly compressed, it is often the case that the image is still not good enough for an operator to distinguish the necessary details. In this case, the use of ROI is an elegant solution to the problem of being able to see the details in part of the image while still maintaining a very high compression level, at the expense of making the other areas in the image less detailed. ROI has been most extensively used in conjunction with medical imaging and are an integral part of

the JPEG-2000 [23] standard. Most ROI techniques are usually used in conjunction with wavelet based image coding techniques [79].

As detailed in chapter 6, for ROI processing there must be a way for the decoder to know which regions were encoded with higher priority than others.

It should be observed that, in general, the use of ROI will impact negatively in the PSNR of the whole reconstructed image but will significantly improve the fidelity in the ROI region itself. In our example, Figure 7.47 shows the difference for coding the region around the text with non linear scaling ROI with 2 common lower bitplanes and 4 upper foreground bitplanes in comparison to the coding the image without ROI. The image used for this is the monochrome 1920×1080 image used in Figure 7.48 and the ROI is a rectangle of dimensions 128×64 with top left corner at $(x_0, y_0) = (1024, 192)$ and bottom right corner at $(x_1, y_1) = (1151, 255)$.

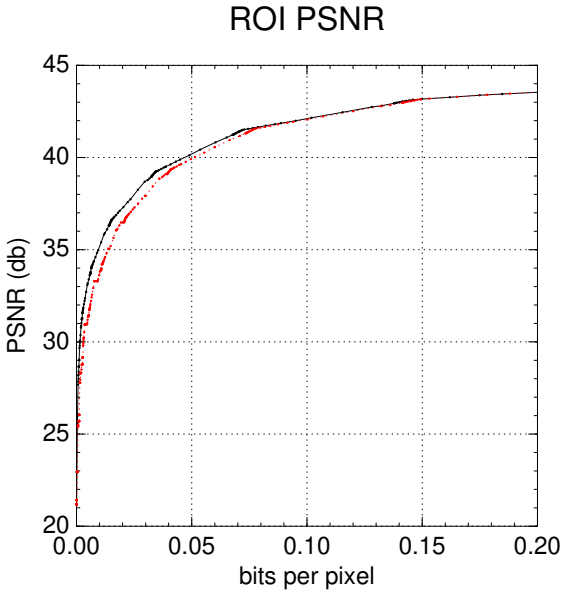


FIGURE 7.47: Effect of ROI in PSNR: without ROI in black and with ROI in red

7.5.1 32-bit ARM Implementation Performance

A working implementation has been developed in the C programming language with special vector code for both Intel and ARM processors to speed up the inner loop of the wavelet transform routine. A few wavelet transforms were implemented: integer wavelet transforms (interpolating biorthogonal integer transforms 5/3, 9/3, 9/7 and 13/7) and the CDF-9/7 real-valued transform. The wavelet used in the next examples

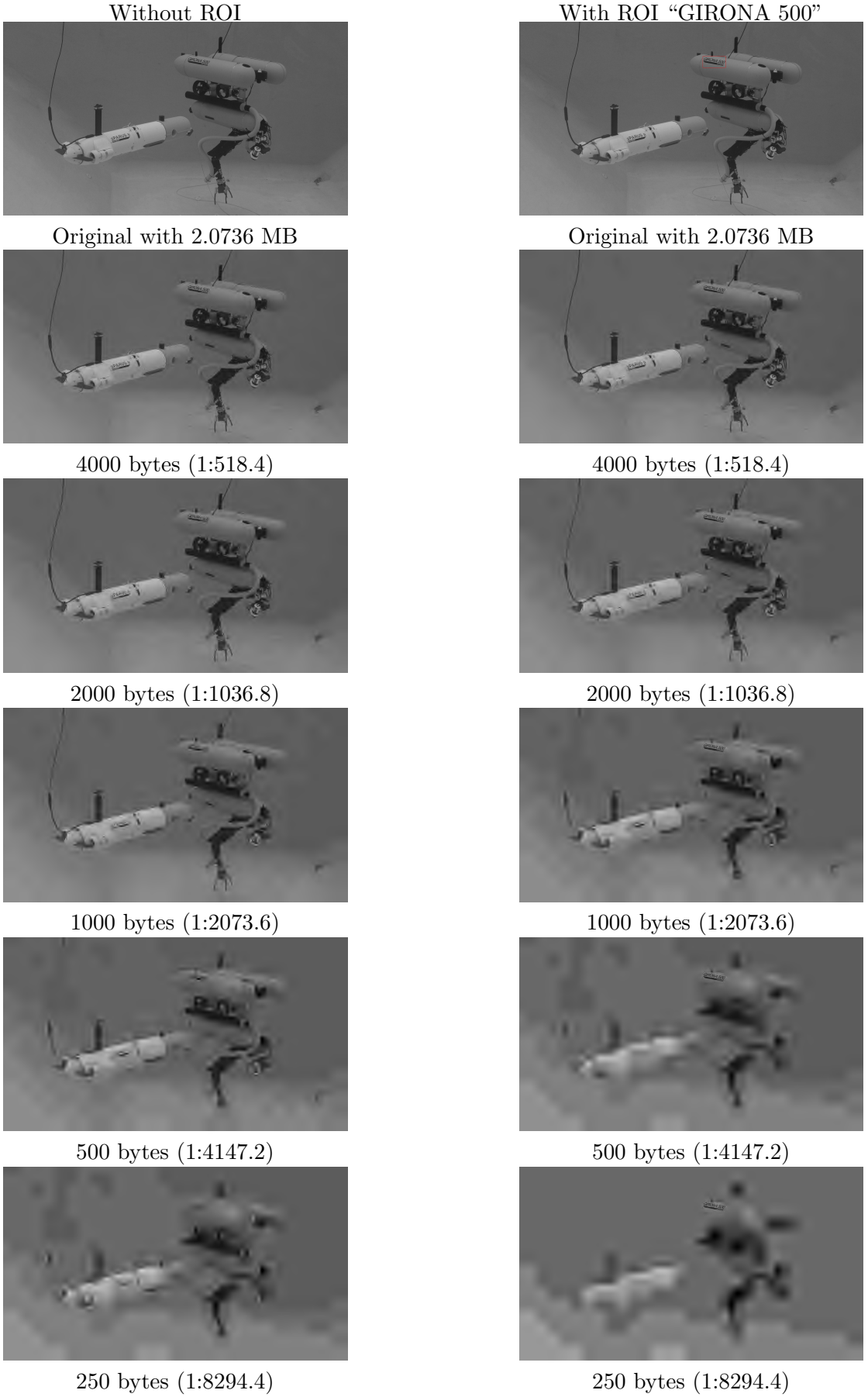


FIGURE 7.48: Comparison of compressed image with and without ROI - DEBT

is the ibior-13/7 transform, which has a highpass filter with 7 taps and a lowpass filter with 13 taps. This wavelet allows for lossless compression and can also be truncated at any point yielding performance within 0.5 db of the CDF-9/7 real-valued transform but being much faster due to the all-integer arithmetic and 16-bit coefficients.

For the 1920×1080 pixels, 8 bpp graylevel image used (Figure 7.48) we ran the compressor in both a Raspberry Pi 2 Model B (RPi2B) and Raspberry Pi 3 Model B (RPi3B). The RPi2B is based on a 1.0 GHz quad code ARM processor (quad core ARM Cortex-A7 with 512KB L2 cache) manufactured by Broadcom (BCM2836 SoC) while the RPi3B, the third generation Raspberry Pi, uses a Broadcom BCM2837 SoC (quad core ARM Cortex-A53 with 512KB of L2 cache) operating at 1.3 GHz with 1 GB of DDR2 RAM. Both use a 32-bit memory bus which was operated at 500 MHz.

The timings for each board for compressing the image on the upper left corner of Fig. 7.48 with and without ROI are displayed on Tables 7.15 and 7.16. The algorithm has a parameter that specifies either the max size or a "quality" factor (which bears some inverse relation with the PSNR). The normal usage (if lossless compression is not required) is to use a "good" quality parameter for local storage and transmission of any prefix of this file for lower quality versions of the compressed image. The "quality" used for each line of the tables was 0, 4, 8, 12, 16, and 24. The wavelet used was the ibior-13/7 b-spline interpolating integer transform with 6 decomposition levels. The first line of each table (where the "quality" parameter is 0) is for the lossless case, where MSE $= 0$ (PSNR $= \infty$) and all timings were based on the single-threaded version of the algorithm.

TABLE 7.15: RPi2B and RPi3B timings (without ROI)

			RPi2B			RPi3B		
Q	Size (bytes)	PSNR (db)	Pre (ms)	Code (ms)	Total (ms)	Pre (ms)	Code (ms)	Total (ms)
0	754628	∞	51.04	307.49	358.54	24.79	204.68	229.48
4	59457	43.00	51.27	37.79	89.07	24.72	26.14	50.87
8	23181	41.18	50.93	14.89	65.83	24.75	10.32	35.07
12	16220	40.27	50.80	10.63	61.43	24.74	7.37	32.12
16	10576	39.13	51.27	6.75	58.03	24.68	4.59	29.27
24	6052	37.37	50.77	3.84	54.61	24.61	2.62	27.23

The column labeled "pre" (tables 7.15 and 7.16) is the time (in milliseconds) for the

TABLE 7.16: RPi2B and RPi3B timings (with ROI)

			RPi2B			RPi3B		
Q	Size (bytes)	PSNR (db)	Pre (ms)	Code (ms)	Total (ms)	Pre (ms)	Code (ms)	Total (ms)
0	754673	∞	83.88	588.50	672.39	41.96	398.17	440.13
4	60278	43.02	83.10	75.41	158.51	42.15	52.37	94.52
8	24609	41.24	83.65	30.91	114.56	41.85	21.57	63.42
12	17797	40.35	83.68	22.90	106.59	41.78	15.93	57.72
16	12429	39.26	83.33	15.35	98.68	42.01	10.49	52.50
24	8050	37.52	83.82	9.63	93.46	41.78	6.53	48.32

wavelet transformation and all other tasks needed to actually start running the compression algorithm (mean extraction, significance map, etc). This step is independent of the amount of bits output and is in fact a lower bound for images of this size (it is almost independent of the contents of the image itself and mostly dependent on the image dimensions alone).

The column labeled "code" (Tables 7.15 and 7.16) is the time (in milliseconds) for the actual compression algorithm and is directly proportional to the amount of bits output. Therefore, the specification of a quality factor or a maximum size will have a great impact on this part and in the total running time for the image compression.

Also, the timings shown represent an upper bound and there is always room for implementation improvement, specially regarding the scaling ROI code.

7.5.2 DEBT × JPEG-2000 Compression Comparison

Table 7.17 compares DEBT with JPEG-2000 [23]. Once again, the JPEG-2000 [23] implementation used was the "`JasPer`" program [78] version 2.0.14 with 6 levels of decompositions and the CDF-9/7 wavelet transform. In order to do a "fair" comparison, we have also included a run of our algorithm with the same number of decompositions (6) and the same wavelet (CDF-9/7) along with the previously used 6 levels of decompositions and the ibior-13/7 integer wavelet transform (used for the timing results in the previous tables).

It is important to note that, because JPEG-2000 [23] does not have an option of exact output size, the amount of bytes used in the comparison was given by the resulting size

Table 7.17: DEBT × JPEG-2000

Rate	Size	PSNR (db)		
	(bytes)	JPEG-2000	DEBT ibior-13/7	DEBT cdf-9/7
0.0001	179	19.15	27.50	27.65
0.0002	400	28.15	30.23	29.94
0.0005	1027	31.87	32.43	32.47
0.001	2055	33.78	34.30	34.43
0.002	4080	35.96	36.25	36.67
0.005	10339	38.60	39.09	39.56
0.01	20702	40.66	40.85	41.68
0.02	41238	42.63	42.27	43.25
0.05	103634	44.60	44.11	44.84
0.1	207315	45.98	45.92	46.25

of the JPEG-2000 [23] file by using the following: `jasper --input tank.pgm --output tank.jpc --output-format jpc -O rate=X`, where `X` is the rate (first column) for the compression. The resulting file size was then used to compress the same image using DEBT to this exact size, once with our current parameters (6 decomposition levels and the ibior-13/7 DWT) and another with the same parameters as the ones used in the JPEG-2000 [23] case (6 decomposition levels and the CDF-9/7 DWT).

The results show that, for the example image used, our algorithm is quite competitive with the current state-of-the-art JPEG-2000 [23] codec, even when using the ibior-13/7 DWT and is vastly superior for very small rates using either DWT. In our example, DEBT constantly outperforms JPEG-2000 [23] when using the same number of decompositions (6) and both transforms, while being much faster.

Once again, the current DEBT algorithm does not employ any final entropy coding step and does not take advantage of any inter band correlation among the coefficients which certainly has a negative impact on the final compression ratio but a positive one on the compression performance (latency). However, even with these design trade-offs it consistently performs better then JPEG-2000 [23] on all tested images at all bit rates and much better at very low bit rates. In the example above, its PSNR is 8.5db higher then JPEG-2000 [23] at the rate 1:10000.

Chapter 8

Conclusions

This chapter presents the contributions of the present work and the associated conclusions regarding the proposed methods and its implementation that contribute towards a solution to the problem of image communications across extremely low bandwidth channels. Suggestions for future research that could complement the present work are also presented.

Appendix D presents the list of published and submitted papers which were generated during the research period.

8.1 Contributions

The main contributions of this thesis were:

1. Minimal time Discrete Wavelet Transform algorithm: The Discrete Wavelet Transform is an essential part of many data analysis procedures and algorithms and its calculation is of paramount importance in many fields of signal processing. The presented paraline algorithm allows implementations to achieve minimal running times and benefits all systems which make use of such a transform. A parallel algorithm has been designed, implemented, and benchmarked which calculates the 2 dimensional DWT in the same time as a simple inplace matrix copy, depending on the number of CPU cores, their performance, and the maximum bandwidth of the memory subsystem.

2. Rate-Distortion optimized ordering for significant and refinement bits: It is shown that each corresponding refinement level (same bitplane) for each class of coefficient significance (Table 4.1) contributes differently to the distortion reduction for an EPD PDF except for an uniform or a laplace PDF (Table 4.2), which may be viewed as special cases of an EPD PDF when the shape parameter $s \to \infty$ and $s = 1$, respectively. By using these weights together with the DWT subband gains, a rate-distortion optimized ordering for the significant and refinement bits is derived which is shown to improve the rate-distortion curves in relation to using a simple predefined uniform or laplace PDF.

3. Fast parallel progressive set partitioning image compression algorithm with ROI: A parallel progressive bit-oriented output stream compression algorithm which excels at low bit rates is presented. The DEBT algorithm makes use of variable depth blocks and trees and is composed essentially by a transformation step followed by set and coefficient testing according to a priority list, without any final entropy coding step. This makes is very flexible and fast in exchange for slightly worse compression ratio when compared to what could be achieved by not omitting this last entropy coding step. However, it still compresses better then current non set-partitioning algorithms for most natural and underwater images tested, for all bit rates, while being much faster and allowing for user defined packet sizes. It can be used in a lossless manner when coupled with an integer reversible DWT and locally store exact representations of the captured images while transmitting any prefix of it in order to cope with latency, bandwidth, or protocol restrictions.

4. General Region Of Interest (ROI) coding with non-linear scaling: Both mapping and non-mapping ROI are implemented which allows for the reduction of source information for extremely low bandwidth communications by selecting regions which are encoded in high quality and blending them with the rest of the image which is encoded with lower quality. Non linear scaling can also be used by delaying the encoding of the lower foreground bitplanes, which consist of a large number of mostly noisy data which improves the context information (background) by a substantial margin. The DEBT algorithm can also efficiently cope with complex ROI geometries, which cannot be compactly described and would be prohibitively expensive to be sent as overhead information, by using exclusive ROI masks with common lower bitplanes and not sending any map information at all at the cost of slightly

worse reconstruction quality. Also, ROI can be used with lossless compression so its use does not interfere with the exact representation stored locally.

5. Available implementation: A DEBT algorithm implementation is available for both ARM (linux 32 and 64 bit) and x86 (linux 32 and 64 bit and MacOS 64 bit) platforms and uses the minimum-time parallel DWT algorithm "paraline", which is also available as static and dynamic libraries for the same platforms.

Extremely low bandwidth communications with user defined packet sizes can be tackled with a tool which is appropriate for the implementation of communications protocols which must embed image information in the communications stream. In high latency or low bandwidth scenarios, the image may be sent along with control data, possibly filling available variable size packets without the need to make any pre-arrangements reserve a minimum space for it.

In fact, it solves the main problems detected in the recent literature (Kaeli [22]) and eases the implementation of flexible protocols by allowing the user to embed any prefix of the compressed image to whatever size is available in order to have the "best" approximation for that size.

8.2 Suggestions and Future Work

There are many possible modifications to the DEBT algorithm that could prove useful in different situations.

Firstly, an investigation regarding the inter band correlation of the transform coefficients should be done in order to compress even further the significance map (address information) without significantly impacting the parallel nature of the algorithm. It is clear that the current DEBT algorithm is not taking advantage of almost any inter band correlation from the fact that using blocks and trees or only using blocks essentially result in similar compression ratios. Some work has been done in a preliminary fashion and it seems that there is not much correlation between the significance of a lower frequency (coarser) subband at the same or at a higher bitplane with the significance of a higher frequency (finer) subband. However, it remains to be investigated if there is a bias when

a higher frequency (finer) subband is significant at a higher bitplane than a lower frequency (coarser) subband and, in case the finer (subband,bitplane) precedes the coarser (subband,bitplane) in the priority list, it might be the case of assigning a shorter code to the same significance partition (this would simply imply that large discontinuities would propagate across scales, e.g., an edge between different objects).

Different strategies for coding the significance information could be easily used with the rate distortion optimized ordering presented in this thesis, for example, Wavelet Reduction Difference (WDR) [80, 81] and many of its variants, like Adaptively Scanned Wavelet Difference Reduction (ASWDR) [82], could be used instead of the variable depth blocks and trees used here. This could make it easier to implement random access on the compressed image.

Also, it might make sense to create a version of the DEBT algorithm with a final entropy coding step, specially for the sign and significance information, which is composed of a sequence of set partitions and cannot withstand transmission errors. While the refinement information may also benefit from entropy coding, it seems that there are little gains to be achieved and, at the same time, would make it susceptible to transmission errors. In all cases, this would have an impact on the performance of the algorithm and its parallel implementation but which may be fast enough for newer embedded platforms while presenting better compression ratios.

Preliminary code profiling shows that, apart from the transformation step which is already optimized with the paraline algorithm, the code that processes the set partition and decomposition (significance steps marked with a “s” in the second column of Table 5.3) is the one that has the most potential to generate speedups. The refinement steps (marked with an “r” in the second column of Table 5.3) are quite fast and represent a small fraction of the execution time of the significance steps (marked with an “s” in the second column of Table 5.3).

Bibliography

[1] F. Arrichiello, D. N. Liu, S. Yerramalli, A. Pereira, J. Das, U. Mitra, and G. S. Sukhatme. Effects of underwater communication constraints on the control of marine robot teams. In *2009 Second International Conference on Robot Communication and Coordination*, pages 1–8, March 2009. doi: 10.4108/ICST.ROBOCOMM2009.5826.

[2] D. Centelles, E. Moscoso, G. Vallicrosa, N. Palomeras, J. Sales, J. V. Mart, R. Marn, P. Ridao, and P. J. Sanz. Wireless hrov control with compressed visual feedback over an acoustic link. In *OCEANS 2017 - Aberdeen*, pages 1–7, June 2017. doi: 10.1109/OCEANSE.2017.8084979.

[3] Eduardo M Rubino, Diego Centelles, Jorge Sales, José V Martı, Raúl Marın, and Pedro J Sanz. Image compression with region of interest for underwater robotic archaeological applications. In *XXXVI Jornadas de Automtica*, Bilbao, Spain, 2015. ISBN 978-84-15914-12-9.

[4] Pedro J. Sanz, Mario Prats, Pere Ridao, David Ribas, Gabriel Oliver, and Alberto Orti. Recent progress in the RAUVI project. A reconfigurable autonomous underwater vehicle for intervention. In *52-th International Symphosium ELMAR-2010*, pages 471–474, Zadar, Croatia, September 2010.

[5] Pedro J. Sanz, Antonio Peñalver, Jorge Sales, David Fornas, José Javier Fernández, Javier Perez, and José Antonio Bernabé. GRASPER: A multisensory based manipulation system for underwater operations. In *2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Manchester, UK, Oct 2013. IEEE.

[6] Pedro J. Sanz, Pere Ridao, Gabriel Oliver, Giuseppe Casalino, Yvan Petillot, Carlos Silvestre, Claudio Melchiorri, and Alessio Turetta. TRIDENT: An european project targeted to increase the autonomy levels for underwater intervention missions. In

OCEANS13 MTS/IEEE conference, pages 1–10, San Diego, CA, Sept 2013. ISBN 978-0-933957-40-4.

[7] E. M. Rubino, J. Sales, D. Centelles, P. J. Sanz, R. Marn, and J. V. Marti. Image compression with Region Of Interest for Underwater Robotic Archaeological Applications. In *XXXVI Jornadas de Automtica)*, Bilbao, Spain, september 2015. ISBN 978-84-15914-12-9.

[8] J. J. Fernndez, J. Prez, A. Pealver, J. Sales, D. Fornas, and P. J. Sanz. Benchmarking using uwsim, simurv and ros: An autonomous free floating dredging intervention case study. In *OCEANS 2015 - Genova*, pages 1–7, May 2015. doi: 10.1109/OCEANS-Genova.2015.7271514.

[9] M.Suzuki and T.Sasaki. Digital acoustic image transmission system for deep sea research submersible. In *in Proc. IEEE Oceans'92 Conference*, pages 567–570, Feb 1992. doi: DOI10.1109/DCC.2016.19.

[10] J. Gomes, V. Barroso, G. Ayela, and P. Coince. An overview of the asimov acoustic communication system. In *OCEANS 2000 MTS/IEEE Conference and Exhibition. Conference Proceedings (Cat. No.00CH37158)*, volume 3, pages 1633–1637 vol.3, 2000. doi: 10.1109/OCEANS.2000.882174.

[11] D. F. Hoag, V. K. Ingle, and R. J. Gaudette. Low-bit-rate coding of underwater video using wavelet-based compression algorithms. *IEEE Journal of Oceanic Engineering*, 22(2):393–400, Apr 1997. ISSN 0364-9059. doi: 10.1109/48.585958.

[12] S. Negahdaripour and A. Khamene. Motion-based compression of underwater video imagery for the operations of unmanned submersible vehicles. *Computer Vision and Image Understanding*, 79(1):162 – 183, 2000. ISSN 1077-3142. doi: http://dx.doi.org/10.1006/cviu.2000.0845. URL http://www.sciencedirect.com/science/article/pii/S1077314200908452.

[13] Konstantinos Pelekanakis. Design and analysis of a high-rate acoustic link for underwater video transmission. In *Thesis, Massachusetts Institute of Technology*, page 75, Jun 2004.

[14] R.L. Eastwood, L.E. Freitag, and J.A. Catipovic. Compression techniques for improving underwater acoustic transmission of images and data. In *OCEANS '96. MTS/IEEE, IEEE Xplore.*, pages 63–68, Sep 2002.

[15] James S Walker, Truong Q Nguyen, and Ying-Jui Chen. A low-power, low-memory system for wavelet-based image compression. *Optical Engineering, Research Signposts*, 5:111–125, 2003.

[16] W.A. Pearlman, A. Islam, N. Nagaraj, and A. Said. Efficient, low-complexity image coding with a set-partitioning embedded block coder. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(11):1219–1235, Nov 2004. ISSN 1051-8215. doi: 10.1109/TCSVT.2004.835150.

[17] Christopher Alden Murphy. Progressively communicating rich telemetry from autonomous underwater vehicles via relays. In *Thesis, Massachusetts Institute of Technology and WOODS HOLE OCEANOGRAPHIC INSTITUTION*, page 75, Jun 2012.

[18] Huanyang Zheng, Ning Wang, and Jie Wu. Minimizing deep sea data collection delay with autonomous underwater vehicles. *Parallel and Distributed Computing*, pages 1–43, January 2017.

[19] Ranjan K. Senapati, Umesh C. Pati, and Kamala Kanta Mahapatra. Listless block-tree set partitioning algorithm for very low bit rate embedded image compression. {*AEU*} - *International Journal of Electronics and Communications*, 66(12):985 – 995, 2012. ISSN 1434-8411. doi: https://doi.org/10.1016/j.aeue.2012.05.001. URL http://www.sciencedirect.com/science/article/pii/S1434841112001070.

[20] Yang Zhang, Shahriar Negahdaripour, and Qingzhong Li. Low bit-rate compression of underwater imagery based on adaptive hybrid wavelets and directional filter banks. *Signal Processing: Image Communication*, 47:96 – 114, 2016. ISSN 0923-5965. doi: http://dx.doi.org/10.1016/j.image.2016.06.001. URL http://www.sciencedirect.com/science/article/pii/S092359651630087X.

[21] Usama S. Mohammed and H. A. Hamada. An efficient rate allocation scheme with optimum peak-to-average power ratio for transmission of image streams over ofdm channels. *International Journal of Video Security*, 10(10):30–42, 2013. ISSN 95610-7474.

[22] Jeffrey W. Kaeli. Computational strategies for understanding underwater optical image datasets. In *Thesis, Massachusetts Institute of Technology*, page 135, Nov 2013.

[23] David S. Taubman and Michael W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, Norwell, MA, USA, 2001. ISBN 079237519X.

[24] W. B. Pennebaker and J. L. Mitchell. JPEG still image data compression standard. New York: Van Nostrand Reinhold, 1992, 1992.

[25] A. Said and W.A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(3):243–250, Jun 1996. ISSN 1051-8215. doi: 10.1109/76.499834.

[26] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *Signal Processing, IEEE Transactions on*, 41(12):3445–3462, Dec 1993. ISSN 1053-587X. doi: 10.1109/78.258085.

[27] Frederick W. Wheeler and W.A. Pearlman. Combined spatial and subband block coding of images. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 3, pages 861–864 vol.3, 2000. doi: 10.1109/ICIP.2000.899592.

[28] A.A. Moinuddin and E. Khan. Wavelet based embedded image coding using unified zero-block-zero-tree approach. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II, May 2006. doi: 10.1109/ICASSP.2006.1660377.

[29] E.S. Hong and R.E. Ladner. Group testing for image compression. *Image Processing, IEEE Transactions on*, 11(8):901–911, Aug 2002. ISSN 1057-7149. doi: 10.1109/TIP.2002.801124.

[30] P. J. Sanz, A. Pealver, J. Sales, J. J. Fernndez, J. Prez, D. Fornas, J.C. Garca, and R. Marin. Multipurpose underwater manipulation for archaeological intervention. In *Sixth International Workshop on Marine Technology (MARTECH'15)*, Cartagena, Spain, Sep 2015.

[31] M. D. Adams and F. Kossentni. Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis. *IEEE Transactions on Image Processing*, 9(6):1010–1024, Jun 2000. ISSN 1057-7149. doi: 10.1109/83.846244.

[32] Stephane Mallat. Elsevier, third edition. In *A Wavelet Tour of Signal Processing. The Sparse Way*, page 805, 2009. ISBN 13: 978-0-12-374370-1.

[33] Hamada Ahmed Hamada Esmaiel. *Advanced Multi-Band Modulation Technology for Underwater Communication Systems*. University of Tasmania, Doctoral Thesis, 2015.

[34] Danchi Jiang Hamada Esmaiel. Optimum bit rate for image transmission over underwater acoustic channel. *Journal of Electrical and Electronic Engineering*, 2 (4):64–74, 2014. ISSN 2329-1613. doi: 10.11648/j.jeee.20140204.12.

[35] Beatrice Tomasi, Laura Toni, Paolo Casari, James Preisig, and Michele Zorzi. A study on the spiht image coding technique for underwater acoustic communications. In *Proceedings of the Sixth ACM International Workshop on Underwater Networks*, WUWNet '11, pages 9:1–9:8, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1151-9. doi: 10.1145/2076569.2076578. URL http://doi.acm.org/10.1145/2076569.2076578.

[36] T. B. Santoso, Wirawan, and G. Hendrantoro. Image transmission with ofdm technique in underwater acoustic environment. In *2012 7th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, pages 37–41, Oct 2012. doi: 10.1109/TSSA.2012.6366017.

[37] R. L. Eastwood, L. E. Freitag, and J. A. Catipovic. Compression techniques for improving underwater acoustic transmission of images and data. In *OCEANS 96 MTS/IEEE Conference Proceedings. The Coastal Ocean - Prospects for the 21st Century*, page 67 suppl., Sep 1996. doi: 10.1109/OCEANS.1996.566719.

[38] Yi Sun, Ran ming Li, and Xiao lei Cao. Image compression method of terrain based on antonini wavelet transform. In *Proceedings. 2005 IEEE International Geoscience and Remote Sensing Symposium, 2005. IGARSS '05.*, volume 2, pages 4 pp.–, July 2005. doi: 10.1109/IGARSS.2005.1525200.

[39] M. J. Weinberger, G. Seroussi, and G. Sapiro. The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls. *IEEE Transactions on Image Processing*, 9(8):1309–1324, Aug 2000. ISSN 1057-7149. doi: 10.1109/83.855427.

[40] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, Apr 1983. ISSN 0090-6778. doi: 10.1109/TCOM.1983.1095851.

[41] M. N. Do and M. Vetterli. Framing pyramids. *IEEE Transactions on Signal Processing*, 51(9):2329–2342, Sept 2003. ISSN 1053-587X. doi: 10.1109/TSP.2003.815389.

[42] Ingrid Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. ISBN 0-89871-274-2.

[43] D. Taubman. High performance scalable image compression with ebcot. *IEEE Transactions on Image Processing*, 9(7):1158–1170, Jul 2000. ISSN 1057-7149. doi: 10.1109/83.847830.

[44] A. Cohen, Ingrid Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5):485–560, 1992. ISSN 1097-0312. doi: 10.1002/cpa.3160450502. URL http://dx.doi.org/10.1002/cpa.3160450502.

[45] A.R. Calderbank, Ingrid Daubechies, Wim Sweldens, and Boon-Lock Yeo. Wavelet transforms that map integers to integers. *Applied and Computational Harmonic Analysis*, 5(3):332 – 369, 1998. ISSN 1063-5203. doi: http://dx.doi.org/10.1006/acha.1997.0238. URL http://www.sciencedirect.com/science/article/pii/S1063520397902384.

[46] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications*, pages 247–269, December 1998. doi: 10.1007/BF02476026.

[47] David Barina. Lifting scheme cores for wavelet transform. In *Thesis, Brno University of Technology, Brno*, page 80, Dec 2015.

[48] Ana Lucia Varbanescu. On the effective parallel programming of multi-core processors. In *Thesis, Universitatea POLITEHNICA Bucuresti, Romania*, page 217, Jun 2010.

[49] Bo-Cheng Charles Lai, Kun-Chun Li, Guan-Ru Li, and Chin-Hsuan Chiang c. Self adaptable multithreaded object detection on embedded multicore systems. *J. Parallel Distrib. Comput.*, pages 25–38, January 2015.

[50] Cheng yi Xionga, Jian hua Houa, Jin wen Tianb, and Jian Liub. Efficient array architectures for multi-dimensional lifting-based discrete wavelet transforms. *Signal Processing*, pages 1089–1099, October 2007.

[51] Ricardo Jose Colom-Palero, Rafael Gadea-Girones, Francisco Jose Ballester-Merelo, and Marcos Martnez-Peiro. Flexible architecture for the implementation of the two-dimensional discrete wavelet transform (2d-dwt) oriented to fpga devices. *Microprocessors and Microsystems*, pages 509–518, June 2004.

[52] David Barina, Ondrej Klima, and Pavel Zemcik. Single-loop software architecture for jpeg 2000. In *Data Compression Conference (DCC)*, pages 1 pp.–, Feb 2016. doi: DOI10.1109/DCC.2016.19.

[53] R. Kutil. A single-loop approach to simd parallelization of 2d wavelet lifting. In *14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06)*, pages 8 pp.–, Feb 2006. doi: 10.1109/PDP.2006.14.

[54] A. Shahbahrami, B. Juurlink, and S. Vassiliadis. Implementing the 2-d wavelet transform on simd-enhanced general-purpose processors. *IEEE Transactions on Multimedia*, 10(1):43–51, Jan 2008. ISSN 1520-9210. doi: 10.1109/TMM.2007.911195.

[55] David Barina, Michal Kula, and Pavel Zemcik. Parallel wavelet schemes for images. *Journal of Real-Time Image Processing*, pages 1–17, 2016. ISSN 1861-8219. doi: 10.1007/s11554-016-0646-3. URL http://dx.doi.org/10.1007/s11554-016-0646-3.

[56] Daniel Chaver, Christian Tenllado, Luis Piñuel, Manuel Prieto, and Francisco Tirado. *2-D Wavelet Transform Enhancement on General- Purpose Microprocessors: Memory Hierarchy and SIMD Parallelism Exploitation*, pages 9–21. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-36265-4. doi: 10.1007/3-540-36265-7_2. URL http://dx.doi.org/10.1007/3-540-36265-7_2.

[57] David Barina and Pavel Zemcik. Vectorization and parallelization of 2-d wavelet lifting. *Journal of Real-Time Image Processing*, pages 1–13, 2015. ISSN 1861-8219. doi: 10.1007/s11554-015-0486-6. URL http://dx.doi.org/10.1007/s11554-015-0486-6.

[58] Daniel Chaver, Christian Tenllado, Luis Piñuel, Manuel Prieto, and Francisco Tirado. *Wavelet Transform for Large Scale Image Processing on Modern Microprocessors*, pages 549–562. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-36569-3. doi: 10.1007/3-540-36569-9_37. URL http://dx.doi.org/10.1007/3-540-36569-9_37.

[59] Andrei C. Jalba Wladimir J. van der Laan and Jos B. T. M. Roerdink. Accelerating wavelet lifting on graphics hardware using cuda. *IEEE Transactions on Parallel and Distributed Systems*, pages 132–146, 2011. ISSN 1045-9219. doi: 10.1109/TPDS.2010.143. URL http://dx.doi.org/10.1007/s11554-015-0486-6.

[60] Takuya Ikuzawa, Fumihiko Ino, and Kenichi Hagihara. Reducing memory usage by the lifting-based discrete wavelet transform with a unified buffer on a {GPU}. *Journal of Parallel and Distributed Computing*, 9394:44 – 55, 2016. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/j.jpdc.2016.03.010. URL http://www.sciencedirect.com/science/article/pii/S0743731516300016.

[61] R. Bozinovic and Z. Markovic. Fast dwt-based intermediate video codec optimized for massively parallel architecture, September 20 2016. URL https://www.google.ch/patents/US9451291. US Patent 9,451,291.

[62] Alex Hutcheson and Vincent Natoli. Memory bound vs. compute bound: A quantitative study of cache and memory bandwidth in high performance applications. Technical report, Stone Ridge Technology, Charlottesville, Virginia, 2011. URL http://stoneridgetechnology.com/wp-content/uploads/2014/12/ComputevsMemory.pdf.

[63] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995.

[64] David Barina, Ondrej Klima, and Pavel Zemcik. Minimum memory vectorisation of wavelet lifting. In *Advanced Concepts for Intel. Vision Systems (ACIVS)*, pages pp.91–101, Feb 2013. doi: DOI10.1109/DCC.2016.19.

[65] S. Chatterjee and C. D. Brooks. Cache-efficient wavelet lifting in jpeg 2000. In *Proceedings. IEEE International Conference on Multimedia and Expo*, volume 1, pages 797–800 vol.1, 2002. doi: 10.1109/ICME.2002.1035902.

[66] D. Barina, O. Klima, and P. Zemcik. Single-loop software architecture for jpeg 2000. In *2016 Data Compression Conference (DCC)*, pages 582–582, March 2016. doi: 10.1109/DCC.2016.19.

[67] John D. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. URL http://www.cs.virginia.edu/stream/. A continually updated technical report. http://www.cs.virginia.edu/stream/.

[68] A. Karp and J.F. Collard. Synchronization of threads in a multithreaded computer program, December 22 2005. URL https://www.google.com/patents/US20050283780. US Patent App. 10/870,721.

[69] Brendam Gregg. Prentice hall, first edition. In *Systems Performance Enterprise and the Cloud*, page 1128, 2014. ISBN 13: 978-0-13-339009-4.

[70] Joshua Ruggiero. Intel corporation, report. In *Measuring Cache and Memory Latency and CPU to Memory Bandwidth For use with Intel Architecture*, page 14, 2008.

[71] Paul E. McKenney. Memory ordering in modern microprocessors, part i. *Linux J.*, 2005(136):2–, August 2005. ISSN 1075-3583. URL http://dl.acm.org/citation.cfm?id=1080072.1080074.

[72] Intel. Intel corporation, manual. In *Intel 64 and IA-32 Architectures Software Developers Manual*, page 4684, 2016.

[73] E. M. Rubino, A. J. Alvares, R. M. Prades, and P. S. Valero. A novel minimum time parallel 2-d discrete wavelet transform algorithm for general purpose processors. In *2017 46th International Conference on Parallel Processing (ICPP)*, pages 553–562, Aug 2017. doi: 10.1109/ICPP.2017.64.

[74] E. Y. Lam. Statistical modelling of the wavelet coefficients with different bases and decomposition levels. *IEE Proceedings - Vision, Image and Signal Processing*, 151 (3):203–206, June 2004. ISSN 1350-245X. doi: 10.1049/ip-vis:20040398.

[75] K. Sharifi and A. Leon-Garcia. Estimation of shape parameter for generalized gaussian distributions in subband decompositions of video. *Circuits and Systems*

for Video Technology, IEEE Transactions on, 5(1):52–56, Feb 1995. ISSN 1051-8215. doi: 10.1109/76.350779.

[76] V. K. Nath and D. Hazarika. Comparison of generalized gaussian and cauchy distributions in modeling of dyadic rearranged 2d dct coefficients. In *2012 3rd National Conference on Emerging Trends and Applications in Computer Science*, pages 89–92, March 2012. doi: 10.1109/NCETACS.2012.6203305.

[77] Kostas Kokkinakis and Asoke K. Nandi. Exponent parameter estimation for generalized gaussian probability density functions with application to speech modeling. *Signal Processing*, 85(9):1852 – 1858, 2005. ISSN 0165-1684. doi: http://dx.doi.org/10.1016/j.sigpro.2005.02.017. URL http://www.sciencedirect.com/science/article/pii/S0165168405001118.

[78] M.D. Adams and F. Kossentini. Jasper: a software-based JPEG-2000 codec implementation. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 2, pages 53–56, Sept 2000.

[79] V. J. Rehna and M. K. Jeya Kumar. Wavelet based image coding schemes: A recent survey. *CoRR*, abs/1209.2515, 2012. URL http://arxiv.org/abs/1209.2515.

[80] Jun Tian and R. O. Wells. A lossy image codec based on index coding. In *Data Compression Conference, 1996. DCC '96. Proceedings*, pages 456–, March 1996. doi: 10.1109/DCC.1996.488388.

[81] Jun Tian and Raymond O. Wells. *Embedded Image Coding Using Wavelet Difference Reduction*, pages 289–301. Springer US, Boston, MA, 2002. ISBN 978-0-306-47043-1. doi: 10.1007/0-306-47043-8_17. URL https://doi.org/10.1007/0-306-47043-8_17.

[82] J. S. Walker and T. Q. Nguyen. Adaptive scanning methods for wavelet difference reduction in lossy image compression. In *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, volume 3, pages 182–185 vol.3, 2000. doi: 10.1109/ICIP.2000.899325.

Appendix A

DWT Subband Gain Tables

Tables A.1, A.2, A.3, and A.4 show the subband weights for the some integer biorthogonal DWT whose filters are described in the form $[N, \tilde{N}]$ where N is the number of vanishing moments of the analyzing high pass filter while $\tilde{N}$ is the number of vanishing moments of the synthesizing high pass filter. It is also usual to use the L/H nomenclature for a filter, where L is the length of the low pass filter and H is the length of the high pass filter. Also, the weights shown in the tables assume that all filters are using the $\langle 1, 2 \rangle$ normalization so that $\langle G_{\text{DC}}, G_{\text{Nyquist}} \rangle = \langle 1, 2 \rangle$.

Table A.5 shows the subband weights for the Cohen-Daubechies-Feauveau CDF-9/7 symmetric, biorthogonal DWT.

In the lifting equations shown below, the superscript notation indicates the iteration number and it is assumed that the original signal $\{x_j\}$ has been split into even $s_j^0 = x_{2j}$ and odd $d_j^0 = x_{2j+1}$ samples (lazy wavelet transform).

A.1 The 5/3 Integer Discrete Wavelet Transform

The lifting equations for the 5/3 integer wavelet transform are given by the predict step

$$d_j^1 = d_j^0 - \left\lfloor \frac{s_j^0 + s_{j+1}^0}{2} + \frac{1}{2} \right\rfloor \tag{A.1}$$

followed by the update step

$$s_j^1 = s_j^0 + \left\lfloor \frac{d_{j-1}^1 + d_j^1}{4} + \frac{1}{2} \right\rfloor \tag{A.2}$$

and its corresponding subband weights for up to 16 decompositions are shown in table A.1.

TABLE A.1: Weights for 5/3 Integer DWT [2,2] $\langle 1, 2 \rangle$

k	LL_k	$HL_k \mid LH_k$	HH_k
1	1.5	1.03832793	0.71875
2	2.75	1.59221745	0.921875
3	5.375	2.91965985	1.5859375
4	10.6875	5.70278263	3.04296875
5	21.34375	11.3367128	6.02148438
6	42.671875	22.6389236	12.0107422
7	85.3359375	45.2605896	24.0053711
8	170.667969	90.5125427	48.0026855
9	341.333984	181.020767	96.0013428
10	682.666992	362.039398	192.000671
11	1365.3335	724.077698	384.000336
12	2730.66675	1448.15491	768.000183
13	5461.3335	2896.30957	1536.00012
14	10922.667	5792.61865	3072.
15	21845.334	11585.2373	6144.
16	43690.668	23170.4746	12288.

A.2 The 9/3 Integer Discrete Wavelet Transform

The lifting equations for the 9/3 integer wavelet transform are given by the predict step

$$d_j^1 = d_j^0 - \left\lfloor \frac{s_j^0 + s_{j+1}^0}{2} + \frac{1}{2} \right\rfloor \tag{A.3}$$

followed by the update step

$$s_j^1 = s_j^0 + \left\lfloor \frac{19(d_{j-1}^1 + d_j^1) - 3(d_{j-2}^1 + d_{j+1}^1)}{64} + \frac{1}{2} \right\rfloor \tag{A.4}$$

and its corresponding subband weights for up to 16 decompositions are shown in table A.2.

TABLE A.2: Weights for 9/3 Integer DWT [2,4] $\langle 1, 2 \rangle$

k	LL_k	$HL_k \mid LH_k$	HH_k
1	1.5	1.03009522	0.707397461
2	2.75	1.55978763	0.88470459
3	5.375	2.84358168	1.50436401
4	10.6875	5.54431629	2.87620544
5	21.34375	11.0165224	5.6861496
6	42.671875	21.9968987	11.3391685
7	85.3359375	43.9757156	22.6617718
8	170.667969	87.9423904	45.3152618
9	341.333984	175.880249	90.6263809
10	682.666992	351.75824	181.250687
11	1365.3335	703.515381	362.500336
12	2730.66675	1407.03015	725.000183
13	5461.3335	2814.06006	1450.00012
14	10922.667	5628.12012	2900.
15	21845.334	11256.2402	5800.
16	43690.668	22512.4805	11600.

A.3 The 9/7 Integer Discrete Wavelet Transform

The lifting equations for the 9/7 integer wavelet transform are given by the predict step

$$d_j^1 = d_j^0 - \left\lfloor \frac{9(s_j^0 + s_{j+1}^0) - (s_{j-1}^0 + s_{j+2}^0)}{16} + \frac{1}{2} \right\rfloor \tag{A.5}$$

followed by the update step

$$s_j^1 = s_j^0 + \left\lfloor \frac{d_{j-1}^1 + d_j^1}{4} + \frac{1}{2} \right\rfloor \tag{A.6}$$

and its corresponding subband weights for up to 16 decompositions are shown in table A.3.

A.4 The 13/7 Integer Discrete Wavelet Transform

The lifting equations for the 13/7 integer wavelet transform are given by the predict step

$$d_j^1 = d_j^0 - \left\lfloor \frac{9(s_j^0 + s_{j+1}^0) - (s_{j-1}^0 + s_{j+2}^0)}{16} + \frac{1}{2} \right\rfloor \tag{A.7}$$

TABLE A.3: Weights for 9/7 Integer DWT [4,2] $\langle 1, 2 \rangle$

k	LL_k	$HL_k \mid LH_k$	HH_k
1	1.640625	1.05104625	0.673339844
2	3.20977783	1.76998842	0.976036072
3	6.40856028	3.46930456	1.87812448
4	12.8155956	6.92767048	3.74486041
5	25.630991	13.8538256	7.48814154
6	51.2619553	27.7074509	14.9760742
7	102.523911	55.4148788	29.9521217
8	205.047821	110.82975	59.9042397
9	410.095642	221.6595	119.808479
10	820.191284	443.319	239.616959
11	1640.38257	886.638	479.233917
12	3280.76514	1773.276	958.467834
13	6561.53027	3546.552	1916.93567
14	13123.0605	7093.104	3833.87134
15	26246.1211	14186.208	7667.74268
16	52492.2422	28372.416	15335.4854

followed by the update step

$$s_j^1 = s_j^0 + \left\lfloor \frac{9(d_{j-1}^1 + d_j^1) - (d_{j-2}^1 + d_{j+1}^1)}{32} + \frac{1}{2} \right\rfloor \tag{A.8}$$

and its corresponding subband weights for up to 16 decompositions are shown in table A.4.

A.5 The CDF-9/7 Discrete Wavelet Transform

The lifting equations for the CDF-9/7 discrete wavelet transform are given by the following successive predict and update steps and final scaling

$$d_j^1 = d_j^0 + \alpha \left(s_j^0 + s_{j+1}^0 \right) \tag{A.9}$$

$$s_j^1 = s_j^0 + \beta \left(d_{j-1}^1 + d_j^1 \right) \tag{A.10}$$

$$d_j^2 = d_j^1 + \gamma \left(s_j^1 + s_{j+1}^1 \right) \tag{A.11}$$

$$s_j^2 = s_j^1 + \delta \left(d_{j-1}^2 + d_j^2 \right) \tag{A.12}$$

$$d_j^3 = \kappa d_j^2 \tag{A.13}$$

TABLE A.4: Weights for 13/7 Integer DWT [4,4] $\langle 1, 2 \rangle$

k	LL_k	$HL_k \mid LH_k$	HH_k
1	1.640625	1.03654814	0.654891968
2	3.20977783	1.73186421	0.934442759
3	6.40856028	3.39070654	1.79398966
4	12.8155956	6.77010059	3.57644415
5	25.630991	13.5386333	7.15128803
6	51.2619553	27.0770607	14.3023653
7	102.523911	54.1540947	28.604702
8	205.047821	108.308182	57.2094002
9	410.095642	216.616364	114.4188
10	820.191284	433.232727	228.837601
11	1640.38257	866.465454	457.675201
12	3280.76514	1732.93091	915.350403
13	6561.53027	3465.86182	1830.70081
14	13123.0605	6931.72363	3661.40161
15	26246.1211	13863.4473	7322.80322
16	52492.2422	27726.8945	14645.6064

$$s_j^3 = \frac{s_j^2}{\kappa} \tag{A.14}$$

where

$$\alpha \approx -1.586134342 \quad \beta \approx -0.05298011857$$

$$\gamma \approx 0.8829110755 \quad \delta \approx 0.4435068520$$

and

$$\kappa \approx 1.230174105$$

and its corresponding subband weights for up to 16 decompositions are shown in table A.5.

TABLE A.5: Weights for CDF-9/7 DWT [4,4] $\langle 1, 2 \rangle$

k	LL_k	$HL_k \mid LH_k$	HH_k
1	1.96590734	1.0112865	0.520217955
2	4.12240982	1.99681246	0.967215776
3	8.41674423	4.18336725	2.07925558
4	16.9355717	8.53411579	4.30048227
5	33.9249268	17.1667252	8.68672371
6	67.8771667	34.3852005	17.418848
7	135.768051	68.7966614	34.8607864
8	271.542969	137.606506	69.7331696
9	543.089355	275.219635	139.472153
10	1086.18042	550.442566	278.947205
11	2172.36182	1100.88672	557.895874
12	4344.72363	2201.77441	1115.79248
13	8689.44824	4403.54883	2231.58521
14	17378.8965	8807.09863	4463.1709
15	34757.793	17614.1973	8926.3418
16	69515.5859	35228.3945	17852.6836

Appendix B

Arbitrary image dimensions

A transformation on either the horizontal or vertical dimension of a previous subband of length n will generate a lowpass subband of length $\lceil \frac{n}{2} \rceil$ and a highpass subband of length $\lfloor \frac{n}{2} \rfloor$. A further decomposition of the lowpass subband of length $k = \lceil \frac{n}{2} \rceil$ will generate a new lowpass subband of length $\lceil \frac{k}{2} \rceil$ and a new highpass subband of length $\lfloor \frac{k}{2} \rfloor$, as seen on Figure B.1.

When n and m are integers, it is easy to see that we can convert from a ceiling operation to a flooring operation by using

$$\left\lceil \frac{n}{m} \right\rceil = \left\lfloor \frac{n+m-1}{m} \right\rfloor \tag{B.1}$$

and that we can convert from a flooring operation to a ceiling operation by using

$$\left\lfloor \frac{n}{m} \right\rfloor = \left\lceil \frac{n-m+1}{m} \right\rceil \tag{B.2}$$

Also, for positive integers n and m and any real number x, a nested ceiling division can be simplified to

$$\left\lceil \frac{\lceil \frac{x}{m} \rceil}{n} \right\rceil = \left\lceil \frac{x}{mn} \right\rceil \tag{B.3}$$

and a nested flooring division can be simplified to

$$\left\lfloor \frac{\lfloor \frac{x}{m} \rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{mn} \right\rfloor \tag{B.4}$$

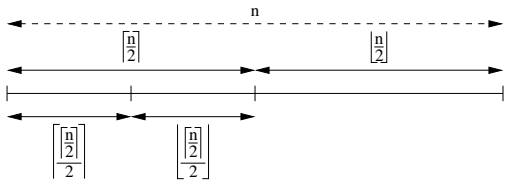


FIGURE B.1: 1 dim transf. (ceil)

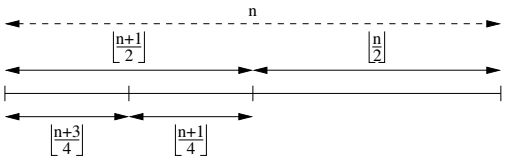


FIGURE B.2: 1 dim transf. (floor)

Figure B.2 depicts the same values where all ceiling operations were converted to flooring operations by using the above identities. By using B.3 it follows that the length of the L_m subband of a dimension of length l after m decompositions is given by $\left\lceil \frac{l}{2^m} \right\rceil$ and the length of the H_m subband is given by $\left\lfloor \frac{\left\lceil \frac{l}{2^{m-1}} \right\rceil}{2} \right\rfloor$ for $m > 0$.

Therefore, any position $0 \leq k < \left\lfloor \frac{n+3}{4} \right\rfloor$ of depth 1 in the L subband will have 2 children at positions $(2k, 2k+1)$ except possibly for the last position $k = \left\lfloor \frac{n+3}{4} \right\rfloor - 1$, which may have either 1 child at position $(2k)$ or 2 children at positions $(2k, 2k+1)$, depending on the parity of $\left\lfloor \frac{n+1}{2} \right\rfloor$ (1 child if odd and 2 children if even), according to B.5.

$$\begin{cases} n \equiv 0 \pmod 4 & \rightarrow (2k, 2k+1) \\ n \equiv 1 \pmod 4 & \rightarrow (2k, 2k+1) \ldots (2k) \\ n \equiv 2 \pmod 4 & \rightarrow (2k, 2k+1) \ldots (2k) \\ n \equiv 3 \pmod 4 & \rightarrow (2k, 2k+1) \end{cases} \tag{B.5}$$

For the H subband the parent child relationship is a little more complicated, depending on the parity of n and $\left\lfloor \frac{n+1}{2} \right\rfloor$. Every position $\left\lfloor \frac{n+3}{4} \right\rfloor \leq k < \left\lfloor \frac{n+1}{2} \right\rfloor$ of depth 1 in the H subband will also have 2 children at either $(2k, 2k+1)$ or $(2k-1, 2k)$ (depending on the parity of $\left\lfloor \frac{n+1}{2} \right\rfloor$) except possibly for the last position $k = \left\lfloor \frac{n+1}{2} \right\rfloor - 1$, which may have 1, 2, or 3 children at positions $(2k)$, $(2k, 2k+1)$ or $(2k-1, 2k)$, and $(2k-1, 2k, 2k+1)$, respectively, according to B.6, i.e., according to the parity of n, the last position is either $2k$ (n odd) or $2k+1$ (n even).

$$\begin{cases} n \equiv 0 \pmod 4 & \rightarrow (2k, 2k+1) \\ n \equiv 1 \pmod 4 & \rightarrow (2k-1, 2k) \\ n \equiv 2 \pmod 4 & \rightarrow (2k-1, 2k) \ldots (2k-1, 2k, 2k+1) \\ n \equiv 3 \pmod 4 & \rightarrow (2k, 2k+1) \ldots (2k) \end{cases} \tag{B.6}$$

It is possible to find the bounds of a rectangle described by its top-left and bottom-right positions $((x_0, y_0), (x_1, y_1))$ at a certain subband by recursively calculating its depth 1 bounds on each dimension using B.5 and B.6. The resulting rectangle will be on the corresponding next lower subband, decreasing the depth by 1.

A decomposition of a LL_m subband will generate four new subbands $\{LL_{m+1}, HL_m, LH_m, HH_m\}$. The bounding box that a position (i, j) on each of the next subbands $\{LL_{m+2}, HL_{m+1}, LH_{m+1}, HH_{m+1}\}$ has on its corresponding subband $\{LL_{m+1}, HL_m, LH_m, HH_m\}$ is the same as the bounding box that a depth 1 block or tree has. By definition, a block or tree belongs to a certain transformation subband if its first block coefficients lie on this subband. Therefore, blocks or trees of depth d rooted in $\{LL_{m+d+1}, HL_{m+d}, LH_{m+d}, HH_{m+d}\}$ will have its first block coefficients at $\{LL_{m+1}, HL_m, LH_m, HH_m\}$. It is possible to find out the bounding box of a position (i, j) of depth d by recursively calculating the bounding boxes of depth 1 on the next subband, until the final subband is reached.

If the position (i, j) is at a horizontal low pass subband, its i-index will use B.5, otherwise, if it is at a horizontal high pass subband, B.6 will be used. Likewise, if the position (i, j) is at a vertical low pass subband, its j-index will use B.5, otherwise, if it is at a vertical high pass subband, B.6 will be used.

Defining a function $g(l, m) = \left\lceil \frac{l}{2^m} \right\rceil$ which describes the resulting low pass length of a dimension of length l (either W for the horizontal or H for the vertical dimension) after applying m decompositions ($m \geq 0$) and its corresponding $g_p(l, m) = g(l, m) \bmod 2$, which describes the parity of $g(l, m)$, along with the number of desired decompositions d, which may be greater than $\lceil \log_2(l) \rceil$, i.e., $d \geq \lceil \log_2(l) \rceil$, the number

$$G(l, r) = g_p(l, r-1)2^{r-1} + g_p(l, r-2)2^{r-2} + \cdots + g_p(l, 1)2 + g_p(l, 0) \tag{B.7}$$

is defined as the low pass generator for this image dimension. It can be seen that

$$G(l, r) = 2^r - l \tag{B.8}$$

and that $g(l, \lceil \log_2(l) \rceil - 1) = 2$ and $g(l, r) = 1$ for $r \geq \lceil \log_2(l) \rceil$. In order to simplify the equations for the descendants, the following function is defined

$$I(l, n, d) = \left\lfloor \frac{G(l, D) \bmod 2^n}{2^{n-d}} \right\rfloor \tag{B.9}$$

where n is the band and d is the depth.

A position i can be at the decomposition subband LL_n if $0 \leq i < \lceil \frac{l}{2^n} \rceil$ and its depth d can range from 0 to n, inclusive, i.e., $0 \leq d \leq n \leq D$. Therefore, a position i with depth d at a dimension of length l (W or H) of the decomposition subband LL_n will have its descendants $\{k\}$ at

$$\begin{cases} 2^d i \leq k < 2^d(i+1) & , 0 \leq i < \lceil \frac{l}{2^n} \rceil - 1 \\ 2^d i \leq k < 2^d(i+1) - I(l, n, d) & , i = \lceil \frac{l}{2^n} \rceil - 1 \end{cases} \tag{B.10}$$

It is easy to see that the last position of the set $\{k\}$ when $i = \lceil \frac{l}{2^n} \rceil - 1$ on B.10 is the length of LL_{n-d} dimension, which is $\lceil \frac{l}{2^{n-d}} \rceil$ and, therefore, we can rewrite B.10 as

$$\begin{cases} 2^d i \leq k < 2^d(i+1) & , 0 \leq i < \lceil \frac{l}{2^n} \rceil - 1 \\ 2^d i \leq k < \lceil \frac{l}{2^{n-d}} \rceil & , i = \lceil \frac{l}{2^n} \rceil - 1 \end{cases} \tag{B.11}$$

We can use B.10 for the L (low pass) dimension of an H_n subband (vertical dimension for the HL_n subband and horizontal dimension for the LH_n subband) but we should note that, in this case, the position i of depth $0 \leq d \leq n < D$ at the L dimension of length l of its corresponding decomposition subband H_n is restricted to $0 \leq i < \lceil \frac{l}{2^{n+1}} \rceil$, i.e., it behaves as if it were an LL band of a dimension of length $\lceil \frac{l}{2} \rceil$. Simply increasing n by 1 in B.10 and B.11 is sufficient to calculate the correct set of descendants. It should also be noted that $I(l, n+1, d) = \left\lfloor \frac{\lfloor \frac{G(l,D)}{2} \rfloor \bmod 2^n}{2^{n-d}} \right\rfloor$.

Therefore, its descendants $\{k\}$ can be calculated according to B.12

$$\begin{cases} 2^d i \leq k < 2^d(i+1) & , 0 \leq i < \lceil \frac{l}{2^{n+1}} \rceil - 1 \\ 2^d i \leq k < 2^d(i+1) - I(l, n+1, d) & , i = \lceil \frac{l}{2^{n+1}} \rceil - 1 \end{cases} \tag{B.12}$$

or

$$\begin{cases} 2^d i \leq k < 2^d(i+1) & ,0 \leq i < \left\lceil \frac{l}{2^{n+1}} \right\rceil - 1 \\ 2^d i \leq k < \left\lceil \frac{l}{2^{n+1-d}} \right\rceil & ,i = \left\lceil \frac{l}{2^{n+1}} \right\rceil - 1 \end{cases} \tag{B.13}$$

It is very important to realize that after a suitable number of decompositions of an image dimension, we are left with only a single coefficient (index 0). While all equations for the L dimensions work naturally in the case where the number of image decompositions D (5.2) exceeds the minimum number of decompositions of this dimension ($\lceil \log_2(W) \rceil$ for the horizontal or $\lceil \log_2(H) \rceil$ for the vertical), we must treat the H dimensions differently if $D \geq \lceil \log_2(l) \rceil$, where l is the length of the current dimension (either W or H). In order to seamlessly use sets (blocks or trees) with any depth $d < D$ for the H dimension, we must make sure that there will always exist a position at index 1 and, to do that, the following function $Q(l, r)$ is defined

$$Q(l, r) = \left\lfloor \frac{2^r - 2^{\lceil \log_2(l) \rceil}}{2} \right\rfloor \tag{B.14}$$

which is the sum of all powers of 2 from $\lceil \log_2(l) \rceil - 1$ up to $r - 1$ (inclusive). With this definition, we can add

$$R(l, n) = \left\lfloor \frac{Q(l, D)}{2^n} \right\rfloor \bmod 2 \tag{B.15}$$

to a position index at a H_n subband in order to always make it 1, in case there are no high pass positions resulting from a decomposition of a previous LL subband of length 2 or less.

For the H (high pass) dimension of an H_n subband (horizontal dimension for the HL_n subband, vertical dimension for the LH_n subband, and horizontal and vertical dimensions for the HH_n subband) a similar scheme may be used by noting that, in this case, defining a function $h(l, m) = \left\lfloor \frac{g(l,m)}{2} \right\rfloor$ which describes the resulting high pass length of a dimension of length l after applying $m + 1$ decompositions ($0 \leq m < \lceil \log_2(l) \rceil$) and its corresponding $h_p(l, m) = h(l, m) \bmod 2$ which describes the parity of $h(l, m)$, the number

$$H(l, r) = Q(l, r) + h_p(l, \lceil \log_2(l) \rceil - 2)2^{\lceil \log_2(l) \rceil - 2} + \cdots + h_p(l, 1)2 + h_p(l, 0) \tag{B.16}$$

is defined as the high pass generator for this image dimension. It can be seen that

$$H(l,r) = G(l, \lceil \log_2(l) \rceil) \oplus \left\lfloor \frac{G(l,r)}{2} \right\rfloor \tag{B.17}$$

where the sum is carried modulo 2, i.e., it is the exclusive-or operation of the binary representation of the 2 integers and that $h(l, \lceil \log_2(l) \rceil - 1) = 1$. Just like it was done before, the following function is defined

$$J(l,n,d) = \left\lfloor \frac{H(l,D) \bmod 2^n}{2^{n-d}} \right\rfloor \tag{B.18}$$

It is now possible to find a general equation for the positions in the H subbands. The set of descendants $\{k\}$ of a position i with depth d at a dimension of length l (W or H) of a high pass subband are

$$\begin{cases} 2^d i \leq k + I(l,n+1,d) < 2^d(i+1) & , \lceil \frac{l}{2^{n+1}} \rceil \leq i < \lceil \frac{l}{2^n} \rceil - 1 \\ 2^d i \leq k + I(l,n+1,d) < 2^d(i+1) - J(l,n,d) & , i = \lceil \frac{l}{2^n} \rceil - 1 + R(l,n) \end{cases} \tag{B.19}$$

or

$$\begin{cases} 2^d i \leq k + I(l,n+1,d) < 2^d(i+1) & , \lceil \frac{l}{2^{n+1}} \rceil \leq i < \lceil \frac{l}{2^n} \rceil - 1 \\ 2^d i - I(l,N+1,d) \leq k < \lceil \frac{l}{2^{n-d}} \rceil + R(l,n-d) & , i = \lceil \frac{l}{2^n} \rceil - 1 + R(l,n) \end{cases} \tag{B.20}$$

Appendix C

Coefficient scaling

Once the standard deviation σ and shape parameter s are known, both can be quantized as $\bar{\sigma}$ and $\bar{s}$, respectively, and sent as side information so that the transform coefficients may be scaled using

$$\bar{x} = x\frac{2^L}{\bar{\sigma}} \tag{C.1}$$

where L is an integer given as

$$L = \lceil \log_2(\bar{\sigma}) \rceil \tag{C.2}$$

simplifying the implementation by using a single table for each quantized shape $\bar{s}$ with binary intervals, i.e., intervals $[2^n, 2^{n+1})$ for integer n. After the scaled coefficients are conveyed, the original values can be retrieved by reversing C.1, i.e.,

$$x = \bar{x}\frac{\bar{\sigma}}{2^L} \tag{C.3}$$

In the lossless case, where the transform coefficients are integers, such a scaling could still be done but, in order to be able to perfectly retrieve the original unscaled values, σ should be quantized to an integer value, e.g., by rounding to the nearest integer

$$\bar{\sigma} = \left\lfloor \sigma + \frac{1}{2} \right\rfloor \tag{C.4}$$

and C.1 becomes

$$\bar{x} = \left\lfloor x\frac{2^L}{\bar{\sigma}} \right\rfloor \tag{C.5}$$

where L is given by C.2 using $\bar{\sigma}$ as given in C.4.

Given that $\bar{\sigma}$ is a positive integer value smaller than or equal to 2^L, i.e., $\bar{\sigma} \leq 2^L$, the original integer transform coefficient x can be retrieved from its expanded value in a similar fashion using

$$x = \left\lceil \bar{x} \frac{\bar{\sigma}}{2^L} \right\rceil \tag{C.6}$$

In general, for positive integers i, m, and n, with $m \leq n$, it can be shown that the integer scaling

$$j = \left\lfloor i \frac{n}{m} \right\rfloor \tag{C.7}$$

is reversible using

$$i = \left\lceil j \frac{m}{n} \right\rceil \tag{C.8}$$

It should be observed that such a scaling could increase the number of bitplanes that need to be conveyed by at most one in certain cases, depending on the value of the maximum absolute transform coefficient and the scaling ratio n/m. It would also increase the time needed for both encoding and decoding by requiring an extra scaling pass for all coefficients. The advantage is the simplification of both the encoder and the decoder by only requiring tables for normalized distributions, i.e., where the standard deviation $\sigma = 1$.

Appendix D

Publications

D.1 Accepted in a Journal

1. RUBINO, E. M. ; ALVARES, A. J.; MARIN, R P ; VALERO, P. S. ; CENTELLES, D. ; SALES, J. ; MARTI, J. V. . Underwater radio frequency image sensor using progressive image compression and region of interest. Journal of the Brazilian Society of Mechanical Sciences and Engineering, v. 39, p. 3259-3279, 2017.

2. RUBINO, E. M. ; ALVARES, A. J. ; MARIN, R P ; SANZ, P. J. ; MARTI, J. V. ; SALES, J. ; CENTELLES, D. . Sistema de Visión subacuático inalámbrico usando un Algoritmo de Compresión progresivo con Región de Interés (SJR 0.27). Revista Iberoamericana de Automatica e Informatica Industrial, 2017.

3. RUBINO, EDUARDO M.; CENTELLES, DIEGO ; SALES, JORGE ; MARTI, JOSE VTE. ; MARIN, RAUL ; SANZ, PEDRO J. . Wireless Image Compression and Transmission for Underwater Robotic Applications. IFAC-PapersOnLine, v. 48, p. 288-293, 2015.

D.2 Submitted to Journal

1. RUBINO, E. M. ; ALVARES, A. J. ; MARIN, R P ; VALERO, P. S. . Real-time Rate Distortion Optimized Image Compression with Region of Interest on the Arm Architecture for Underwater Robotics Applications (SJR 0.31). Journal of Real-Time Image Processing, 2017.

2. RUBINO, E. M. ; ALVARES, A. J. ; MARIN, R P ; VALERO, P. S. . A General Scheme for Finding the Static Rate-Distortion Optimized Ordering for the Bits of the Coefficients for All Subbands of an N-Level Dyadic Biorthogonal DWT (SJR 0.68). SIGNAL PROCESSING-IMAGE COMMUNICATION, 2017.

D.3 IEEE Conferences and Others

1. RUBINO, E. M. ; ALVARES, A. J. ; MARIN, R P ; VALERO, P. S. . A Novel Minimum Time Parallel 2-D Discrete Wavelet Transform Algorithm for General Purpose Processors (SJR 0.27). IEEE Conference Publications, ICPP, Bristol, England, v. 1, p. 553-562, 2017.

2. RUBINO, E. M. ; ALVARES, A. J.; MARIN, R P ; SANZ, P. J. ; CENTELLES, D. ; SALES, J. ; MARTI, J. V. . Progressive Image Compression and Transmission with Region of Interest in Underwater Robotics (SJR 0,13). IEEE Conference Publications, OCEANS 2017, Aberdeen, England, v. 1, p. 1-9, 2017.

3. RUBINO, E. M, Diego Centelles, Guillem Vallicrosa, Narcís Palomeras, Jorge Sales, J. Vicente Martí, Raúl Marín, Pere Ridao, Pedro J.Sanz. Wireless HROV Control with Compressed Visual Feedback over an Acoustic Link, OCEANS 2017, Aberdeen, Regne Unit, 2017.

4. RUBINO, E. M., Diego Centelles, Guillem Vallicrosa, Narcís Palomeras, Jorge Sales, J. Vicente Martí, Raúl Marín, Pere Ridao, Pedro J.Sanz. Control Inalámbrico de un HROV con Realimentación Visual Comprimida, VII jornadas de automática marítima (AUTOMAR), Castelló de la Plana, Espanya, AUTOMAR, UJI (IRS-Lab), 2017.

5. RUBINO, E. M.; CENTELLES, D. ; SALES, J. ; MARTI, J. V. ; MARIN, R. ; SANZ, P. J. . Image compression with Region Of Interest for Underwater Robotic Archaeological Applications. In: XXXVI Jornadas de Automática, 2015, Bilbao. XXXVI Jornadas de Automática. Madri: Comité Español de Automática de la IFAC, 2015. v. 1. p. 856-863.

6. 2. RUBINO, E. M.; CENTELLES, D. ; SALES, J. ; MARTI, J. V. ; MARIN, R. ; SANZ, P. J. . Wireless RF Camera Monitoring for Underwater Cooperative

Robotic Archaeological Applications. In: SIXTH INTERNATIONAL WORKSHOP ON MARINE TECHNOLOGY, 2015, Cartagena. MARINE TECHNOLOGY. Cartagena: Martech, 2015. v. 1. p. 100-102.

7. RUBINO, E. M.; CENTELLES, D. ; SALES, J. ; MARTI, J. V. ; MARIN, R. ; SANZ, P. J. . Wireless Image Compression and Transmission for Underwater Robotic Applications. In: IFAC Workshop Navigation,Guidance and Control of Underwater Vehicles, 2015, Girona. IFAC Workshop Navigation,Guidance and Control of Underwater Vehicles. Londres: IFAC, 2015. v. 1. p. 288-293.

8. RUBINO, E. ; CENTELLES, D.; SOLER, M. ; MARTI, J. V. ; SALES, J. ; MARIN, R. ; SANZ, P. J. . Underwater radio frequency based localization and image transmission system, including specific compression techniques, for autonomous manipulation. In: OCEANS 2015 Genova, 2015, Genova. OCEANS 2015 - Genova, 2015. p. 1.

9. RUBINO, E. M.; MARIN, R. ; SALES, J. ; SANZ, P. J. . Survey on Progressive Image Compression and Transmission and its application in Underwater Intervention Missions. In: XXXV Jornadas de Automática, 2014, Valencia. XXXV Jornadas de Automática. Madri: Comité Español de Automática de la IFAC, 2014. v. 1. p. 10-15.